POLITECNICO DI TORINO

SCUOLA DI DOTTORATO
Dottorato in Ingegneria Elettronica e delle Comunicazioni
XXIV Ciclo

Tesi di Dottorato

Design of Algorithms and Protocols for Peer-To-Peer Streaming Systems

Stefano Traverso
Mtr. 160417

Tutore
prof. Marco Mellia

Gennaio 2012

Summary

Peer-to-Peer Streaming (P2P-TV) systems have been studied in the literature for some time and they are becoming popular among users as well. P2P-TV systems target the real time delivery of a video stream, therefore posing different challenges compared to more traditional peer-to-peer applications such as file sharing (BitTorrent) or VoIP (Skype).

This document focuses on mesh based P2P-TV systems in which the peers form a generic overlay topology at application level upon which peers exchange small "chunks" of video. In particular, we study two problems related with this kind of systems:

i) how to induce peers to share their available resources – such as their available upload bandwidth – in a totally automatic and distributed way;

ii) how to localize P2P-TV traffic in order to lower the load on the underlying transport network without impairing the quality of experience (QoE) perceived by users.

Goal *i)* can be achieved playing on two key aspects of P2P-TV systems that are:

- the design of the trading phase needed to exchange chunks among neighbors;
- the strategy adopted by peers to choose the neighbors to connect with, i.e., the policy employed to build and maintain the overlay topology at application level.

The former task has been successfully accomplished with the development of algorithms that aim at adapting the rate at which peers offer chunks to their neighbors to both peer's available upload bandwidth and to the system demand. The results presented in this document show that the automatic adjustment of transmission rate to available upload capacity reduce delivery delays of chunks, thus improving the experience of users.

Focusing on the latter problem, we prove that the topological properties of the overlay have a deep effect on both users' QoE and network impact. We developed

a smart, flexible and fully distributed algorithm for neighbors selection and implemented it in a real P2P-TV client. This let us compare several different strategies for overlay construction in a large campaign of test-bed experiments. Results show that we can actually achieve the goal of leading peers to efficiently share their available resources – goal *i)* – while keeping a good degree of traffic localization, hence lowering the load on the underlying network – goal *ii)*. Furthermore, our experimental results show that a proper selection of the neighborhood leads to a win-win situation where the performance of the application and QoE are both improved, while the network stress is nicely reduced.

Acknowledgements

I would like to thank my advisor Prof. Marco Mellia together with Prof. Emilio Leonardi and the whole telecommunication group at the Politecnico di Torino: I both learnt a lot and had fun working with you.

I also would to thank Csaba Kiraly, Prof. Renato Lo Cigno and Prof. Luca Abeni from the Department of Computer Science of University of Trento for the great work on P2P-TV client PeerStreamer.

Furthermore I'm thankful to Nikos Laoutaris, Vijay Erramilli and the Internet scientific group of the Telefonica I+D center in Barcelona. Thanks for welcoming me during the period I spent there.

Finally I thank my family, my father and mother, my aunt and my future wife, Lorena, for giving all their support to me without which I would have not been able to reach this goal.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

During the last years we assisted to the growth of interest from the Internet community for Peer-to-Peer streaming (P2P-TV in short) systems. Indeed, these systems offer to broadcasters and providers the opportunity of transmitting TV signals to huge populations of users without expensive infrastructural investments. A clear proof of this interest is given by the continuous appearing of many commercial applications that are nowadays available over the Internet [2–4].

P2P-TV systems are generically based on an *overlay topology* which is built at application level by the peers connecting to each other through logical links. In this kind of systems a *source* peer takes a video-stream, slices it in small pieces called *chunks* and starts injecting them inside the overlay topology. Exploiting a fully distributed epidemic approach, peers act as repeaters to exchange chunks while the video is played. Even if the mechanism behind P2P-TV systems looks similar to P2P file sharing ones, in former systems chunks are generated sequentially, periodically and, overall, in real-time. Chunks must be received by the peers within a *deadline* to be played out and this makes delivery delays one of the key aspects of these systems. For this reason, P2P-TV systems design is deeply different from file sharing applications.

Due to their real-time constraints, P2P streaming systems generate un-elastic traffic. Combining this with the fact that they may reach a potentially unlimited population of users, eventually offering high definition signal (HD-TV) quality streams, P2P-TV systems may constitute a worry for network carriers since the traffic they generate may dramatically grow without control, causing a degradation of the quality of service perceived by Internet users or even the network collapse (and the consequent failure of the P2P-TV service itself!). The goal of this thesis is to provide some simple guidelines for the design of new generation *network-aware* P2P applications that optimize the chunk distribution effectively exploiting some knowledge on the underlay transport network status.

The objective of this work is two-fold: i) we face with problem of inducing peers

to contribute to the chunk diffusion process in a way proportional to their available resources; ii) we focus on the problem of reducing the load of P2P-TV systems on the underlying transport network.

The first goal is successfully achieved by carefully designing the algorithm which handles the trading phase needed before each chunk transmission. This is a critical step to reach a better usage of peers' available bandwidth. The latter objective, instead, is translated into the problem of keeping a certain degree of localization on traffic generated by P2P-TV applications. This can be efficiently sorted out by finely organizing the overlay network which is built by peers at application level. Moreover, acting on the overlay construction process, we can reach together goals i) and ii) in such a way that peers fairly share their available resources while keeping a good degree of traffic localization.

Finally, we show through experimental results that chasing goals i) and ii) does not impair the good quality of received video, but, on the contrary, the QoE perceived by the users is greatly enhanced.

1.1 Transmission Control Problem

The core of chunk distribution algorithms is the chunk scheduling policy, according to which the peers choose which chunks should be delivered to which peers. In the literature, there are two families of algorithms for practically implementing the chosen policy. The *push* based algorithms organize the peers in distribution trees which are rather static and over which a number of consecutive chunks are delivered; in *pull* based approaches, peers are organized in a generic overlay network and a preliminary trading phase is required before the actual chunk delivery: a peer advertises to some of its neighbors which chunks it possesses via the so called *offer* messages and the neighbors, in their turn, choose the chunks they want to receive through *select* messages. By avoiding the trading phase, push based algorithms typically achieve smaller chunk delivery delay than pull based approaches. The drawback is the higher complexity to manage the structure of the tree and a lower robustness to churning (peers moving into or out of the overlay network), which limits their scalability in terms of number of peers. Conversely, in pull based algorithms a careful design of the trading phase is needed to avoid that the additional signaling delay translates into an excessive cost to pay for better resource usage and resilience to churning.

Part I of this thesis presents two different algorithms to adapt the rate at which offer messages are sent by peers to match their available upload bandwidth and the system demand. In other words, both the schemes aim at controlling the bandwidth allocation on the peer uplink channel.

Even if they chase the same objective, the two algorithms deeply differ from

each other: the *Adaptive Signalling Protocol* (ASP) described in Chapter 3 is based on a synchronous mechanism that periodically generates advertisement of peer's buffermaps, which are offered to a subset of the peer's neighbors. Simulation results show that ASP actually reaches the goal of adjusting the chunk transmission rate of peers while improving QoE perceived by users, but the resulting scheme assumes essentially homogeneous latencies and may be difficult to implement in practice.

The latter scheme described in Chapter 4 is called *Hose Rate Control* (HRC) and it aims at automatically adapting the offering rate of peers to the current network scenario simply controlling the queueing delay experienced by delivered chunks. HRC's transmission rate control works in a peer aggregate fashion that aims at exploiting the peer upload capacity while reducing as much as possible chunk delivery delays. Being much simpler to develop with respect to ASP, HRC's control scheme has been implemented in a real P2P-TV application and it has been extensively put on trial in test-bed experiments involving large populations of peers.

1.2 Overlay Topology Design

In Part II we focus on the study of smart algorithms for the construction and management of mesh-based overlay topologies.

In particular Chapter 7 focuses on some issues about unstructured P2P-TV systems. First of all, we estimate what is the real impact of latencies and peer access bandwidth heterogeneity on the performance of a P2P-TV system. Then, we investigate how the application can retrieve measurements coming from the network layer and translate them in useful information to exploit at application level. Then, we analyze the problem of implementing smart algorithms that explicitly exploits the knowledge of network metrics and we study how the application can exploit them. Finally, we come to our final scope: we face with the problem of reducing the load on the underlying transport network while jointly improving the QoE perceived by users. The simple heuristic we propose in Chapter 7 has been tested through simulations and, since it has been integrated in a real P2P-TV application, through experimental test-beds as well. Presented results show that the two objectives of improving the system performance (QoE) and lowering the network stress can be successfully reached at the same time.

Given the useful guidelines provided in Chapter 7, we present in Chapter 8 a fully distributed algorithm for the overlay topology construction and maintenance. The proposed algorithm allows us to combine different neighbor filtering schemes whose weighted choices mechanism consider network level measurements like round trip time (RTT), available bandwidth, path losses, etc. and application level indexes such as amount of received chunks and chunk corruptions. The algorithm has been implemented in a real P2P-TV application and deeply tested in a long campaign

of test-bed experiments. In our evaluation we compare 12 different combinations of filters and prove through a fully experimental evaluation that we can increase the localization of traffic, reducing therefore the load on the network layer. The good news is that combining schemes able to localize the traffic with bandwidth-aware scheme, we can confirm through a rich and rigorous experimental analysis what shown in Chapter 7: we can jointly improve the QoE perceived by users while lowering the network stress caused by the P2P-TV service.

Part I

Transmission Control Problem

Chapter 2

Introduction

In P2P-TV systems, download rate is dictated by video rate, which is limited by definition; the source peer emits chunks in real time at "constant" rate and peers must trade them minimizing delays and losses to guarantee the best Quality of Experience (QoE) to users.

Common assumptions about P2P-TV systems are that i) the upload capacity of peers constitutes the main bottleneck to system performance, and ii) each peer is supposed to instantaneously have a perfect view of the internal state of other peers [5–7]. While the former assumption is often met in practice, latency between peers makes the latter unrealistic [8].

In the literature we can find two different architectural models to implement a real P2P-TV system. The first one relies on a *push* based algorithm for chunk transmissions. This implies that peers are organized in overlay topologies similar to distribution trees which results to be rather static and over which a number of consecutive chunks are delivered (as depicted in the left plot of Figure 2.1). The second approach allows peers to be organized in generic mesh overlay topologies, making the chunk distribution process based on a receiver-driven protocol, also known as *pull* approach (right plot of Figure 2.1). In this case, indeed, before each chunk delivery, a preliminary trading phase is required. During this, a peer advertises to some of its neighbors which chunks it possesses and the neighbors, in their turn, select chunks they are interested in. Thanks to the reduced amount of signalling, typically push based algorithms present smaller chunk delivery delay than pull based approaches. The drawback is the higher complexity to manage the trees and a lower robustness to churning, which limits their scalability in terms of number of peers.

Conversely, also to avoid chunk duplications at the receiver, in pull (also known as receiver-driven) systems a preliminary trading phase is required to agree on the chunks to be exchanged. This trading phase requires the exchange of messages between peer pairs and must be carefully designed to avoid that the additional

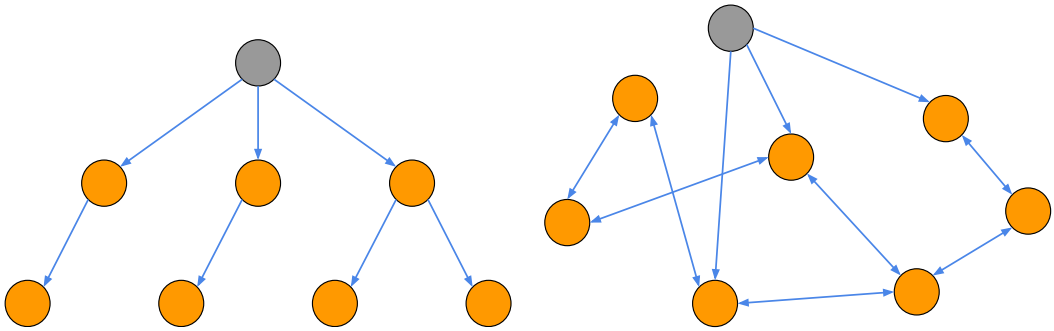

Figure 2.1. Example of highly structured overlay topology built as a distribution tree (left) and example of a generic mesh-based overlay topology (right).

signalling delay translates into an excessive delay and that a peer overbooks its upload capacity by committing itself to transmit of too many chunks.

This part of the thesis focuses on the design of the trading phase, which has been marginally studied in the literature. In general, the trading phase is composed by four messages: *i)* each peer advertises to a subset of its neighbors the set of chunks it possesses through an *offer* message; *ii)* neighboring peers reply to it with a *select* message in which they specify the subset of chunks they are interested in; *iii)* the transmitter then schedules the transmission of the selected chunks using a first in first out (FIFO) queue, from which chunks are served one after the other, since transmitting chunks in sequential order reduces the chunk delivery time with respect to parallel transmissions [9]; *iv)* finally, successfully received chunks are acknowledged to transmitters through an acknowledgement (ACK) message.

This receiver-driven mechanism requires a number of parameters to be tuned to reach optimal results, and the optimal setup, in its turn, depends on the specific scenario, which is typically highly variable due to the natural network variability and user heterogeneity.

In addition, video chunks must be small, e.g., less than 8 packets, to minimize the packetization delay at the source, the transmission delay on the network, and the store-and-forward delay at the peers. To avoid both the burdening of handling TCP, and unnecessary delay due to TCP retransmission and congestion control, UDP is typically preferred by actual P2P-TV application [10]. This poses the problem of how to handle the congestion control, and in particular, how to limit the amount of information a peer can transmit, since its download rate is in all cases limited by the stream rate. Controlling therefore the uplink bandwidth utilization is a key problem, which has been so far practically ignored by the research community.

Several proposals and actual implementations adopt pull based mechanisms (see [8, 9, 11, 12] for example), but, to the best of our knowledge, no discussion about the tuning of the signaling mechanisms have ever been carried on. In this

part, we propose two schemes to automatically adapt the choice of: i) the frequency with which a peer advertises the chunks it possesses, ii) the number of peers to which the chunks are advertised. These two aspects are particularly critical since an inadequate setting can translate into performance degradation, due to excessive signaling overhead, waste of resources or queuing delay at the transmitting peer. We thus propose two solutions to adapt the above mentioned parameters i) to the video rate and ii) to the upload available bandwidth of the peers with the objective of jointly maximizing the exploitation of peers' upload heterogeneity and reducing chunk delivery delay and losses. In other words, the two algorithms carefully control the bandwidth allocation on the uplink channel of a peer.

In Chapters 3 and 4 we present two different algorithms respectively called **Adaptive Signalling Protocol** (ASP) and **Hose Rate Control** (HRC). Even if behind the two schemes we can find almost the same rationale, ASP and HRC deeply differ in the timings at which offer and select messages are handled and sent. The former scheme regulates the rate at which offer messages are sent for a signalling protocol in which offer messages are sent in a synchronous fashion and in small bursts. HRC, instead, relies on a protocol in which offer messages are handled in an asynchronous way that results much more flexible and easier to implement. Both of the schemes have been extensively tested by simulations, but only HRC has been actually integrated into the P2P-TV application called *PeerStreamer*[1] [14]. The results show that, with respect to not adaptive mechanisms, both schemes can consistently improve system performance in terms of chunk loss, delivery delay, and thus QoE. Furthermore, peers' upload capacity is used in such a way that system demand rate is satisfied in a fairer fashion, avoiding stressing low capacity peers, and avoiding concentrating the download from high capacity peers only.

2.1 Related Work

The literature on P2P-TV streaming systems is rich of works. Common assumptions are that the upload capacity of peers constitutes the main bottleneck to system performance, and each peer instantaneously has a perfect view of the internal state of other peers [5–7, 15]. While the former assumption is often met in reality, latency among peers makes the latter unfeasible. The impact of latency on system performance has been analyzed by means of a simple model corroborated by real measurements in PlanetLab in [8]. The authors propose a system that mitigates the effect of latency by exchanging state information via signalling messages. Little description is however given about the implemented signalling mechanisms details.

Few papers specifically focus on the impact of peers bandwidth heterogeneity and how it can be exploited to improve system performance [9, 15, 16]. This aspect

[1] *PeerStreamer* has been developed within the EU-FP7 NAPA-WINE STREP project [13]

is crucial since peers homogeneity is hardly met in practice.

Given the small size of chunks, UDP is typically preferred by actual P2P-TV applications [10] to enforce serial chunk packet transmission and to avoid both unnecessary delay due to TCP retransmission and congestion control. Controlling the uplink bandwidth utilization is thus a key problem.

2.2 System Description

We consider a system in which a source segments the video stream into chunks and injects them in the overlay network. Let $\mathcal{N}$ be the set of peers composing the overlay, with cardinality N. The application must deliver every generated chunk within a deadline starting from the instant in which it is emitted by the source; this deadline is called *playout delay*, D_{max}. If the chunk age is greater then the playout delay, the chunk cannot be traded anymore, as in a sliding window mechanism.

Chunks are transmitted by peers to their neighbors, i.e., they exchange chunks in a swarm-like fashion; the overlay topology is defined by the set of peers and virtual links connecting them. Let $\mathcal{C}_a$ be the set of a neighbors. The overlay topology changes its structure dynamically due to the churning and the possibly dynamic algorithms driving its maintenance and optimization [17]. Since the overlay dynamics are usually much slower than chunk distribution timings (minutes versus tens/hundreds of ms), we are going to neglect churning effects in this part. The overlay can be built by assigning a certain set of neighbors to every peer a. Since the actual design of the overlay topology is handled in Part II, in this part of the document we consider the simplest case in which the overlay network is built once and on a random basis (as also did in [15]).

As normally assumed in the literature of P2P-TV systems, we consider a case where peer's uplink capacity represents the bottleneck to system performance, and consider the chunk delivery loss as main performance indexes (this includes losses and chunks arrived after the playout deadline). In addition, we consider each peer uplink bandwidth utilization as an important index, which allows us to gauge the fairness and efficiency in allocating system upload capacity.

Chapter 3

Adaptive Signalling Protocol

3.1 Introduction

In this chapter we propose a signaling mechanisms that must be in place to trade chunks, combined with an algorithm for automatically adapting a peer service rate to its upload capacity and to match the demand from other peers. The goal is to maximize peer upload capacity utilization, while avoiding forming long transmission queues, therefore minimizing the chunk delivery time which is a crucial parameter for P2P-TV systems.

The objective is to design a an efficient receiver-driven protocol that integrates a fully distributed scheme to automatically tune i) the frequency at which a peer offers chunks to its neighbors and ii) the number of neighbors to contact to offer chunks.

3.2 The Chunk Trading Mechanism

The signaling mechanism used to trade chunk is a *pull* mechanism similar to the one used in other mesh-based P2P-TV systems, [8,9,11,12]. A chunk is sent from a peer to one of its neighbors after a trading phase which is sketched in Fig. 4.1. In the figure, trading messages are represented above the time line and chunk transmissions are below it. The negotiation begins on the transmitter side: peer a periodically chooses a subset of its neighbors $\mathcal{N}_a$ (with $|\mathcal{N}_a| = N_a$) and sends them a special signaling message, called an *offer* message, containing the set of chunks it possesses and whose age is smaller than D_{max}. Every peer in $\mathcal{N}_a$ replies to the offer with a *select* message in which it indicates a subset of at most M desired chunks. Once a chunk has been "selected", the receiver will set it as *pending* to avoid requesting the

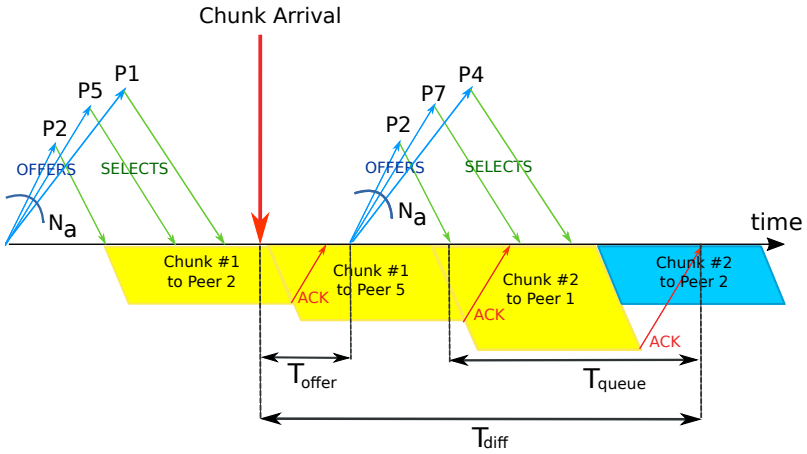

Figure 3.1. Schematic representation of the peer chunk trading mechanism.

same chunk from different peers at the same time[1].

As soon as a receives some *positive select* messages[2], it schedules the transmission of all requested chunks, maintaining a *transmission queue* of chunks to be sent that is served in a FIFO order. Peer a is committed to send all the chunks requested in all the received select messages. Chunks are small, to minimize the transmission delay and to quickly spread them through the overlay via the store-and-forward mechanism typical of the P2P systems.

Several design choices impact the performance of the pull mechanism: 1) the criterion to select peers belonging to $\mathcal{N}_a$ – known as the "peer selection"; 2) the strategy according to which peers in $\mathcal{N}_a$ select chunks to download – known as the "chunk selection"; 3) the frequency at which a peer a offers chunks to its neighbors; and 4) the values of the parameters M and N_a.

Since the objective of our study is to discuss the last two issues, for the *peer selection* and the *chunk selection* policies we make the simplest possible choices: peer a chooses the neighbors to contact uniformly at random within the set of its neighbors, and the neighbors choose the chunks to select at random among the ones it needs. This policy is also known in the literature as "Random Peer - Random Useful Chunk selection" [7].

To keep the chunk delivery delay as low as possible, the length in chunks of the transmission queue must be kept as small as possible; this suggests to: i) set $M = 1$ to avoid filling the transmission queue with many chunks directed to the same neighbor, and ii) issue a new offer based on number of chunks waiting to be transmitted.

[1]Note that *pending* chunks can not be published in *offer* messages yet.

[2]Positive means that at least one chunk was requested in the select message. In this case we assume signaling messages are reliably delivered, e.g., an ARQ mechanism is present.

In next section we describe the algorithm, called ASP (Adaptive Signaling Protocol), we propose to automatically set N_a and decide the schedule of the offer messages.

3.3 The Adaptive Signaling Protocol

Consider a traditional sliding window algorithm adopted to perform congestion control in a end-to-end connection. It is well known that the transmitter window size has to be correctly set to match the actual available bandwidth and RTT. In our P2P-TV peer design, we have to decide the amount of information a peer can transmit to exploit its upload capacity. N_a is equivalent to the transmitter window, measured in chunks, which has to be correctly tuned to match peer a upload capacity, the actual system demand, and the RTT experienced between a and its neighbors. Differently from traditional congestion control algorithms, the overall system upload capacity has to be allocated to match the total download demand rate, since each peer has to contribute to the video distribution and each peer has to download an average amount of information equal to the video rate. Therefore, N_a determines also the bandwidth allocation among peers in the system.

Selecting N_a is not easy. If N_a is too small, a upload bandwidth risks not to be exploited at best: the transmission queue empties quickly, causing long periods of inactivity (*idle times*); this can reduce system performance especially for high upload capacity peers. If, instead, N_a is too large, a transmission queue fills up, causing additional chunk delivery delay and, possibly, losses due to late delivery of chunks. Moreover, a lot of signaling overhead is produced. Thus, N_a must be adapted to the upload capacity of each peer, the average RTT, and the actual system demand rate. Starting from a default value, each peer modifies N_a according to the following algorithm:

if $Tdiff$ -eq $2AvgRTT$ **then**
  $Na \Leftarrow Na + 1$
else
  if $PosSelectNum/OfferNum$ -geq CR **then**
    $Na \Leftarrow Na - 1$
  end if
end if

where

- T_{diff} is the time between a new chunk arrival and the moment in which the transmission queue becomes empty;
- $AvgRTT$ is the round trip time averaged among all peer's neighbors;

- $PosSelectNum$ and $OfferNum$ are respectively the number of received positive select messages and the number of offered messages sent for a given offer/select phase.

- CR is the *clipping ratio* that limits the growth of N_a when the number of positive select messages is small.

The algorithm is run every time a new chunk arrives and only once per trading phase. N_a is thus updated just before sending the offer messages. The algorithm aims at jointly using the available upload bandwidth and maintaining the queue as short as possible. If the transmission queue grows too long, the peer reduces the number of offer messages it sends. On the contrary, if the queue is too short (possible idle times and unused bandwidth), the number of neighbors to contact is increased. The decision is based on T_{diff} that, as sketched in Fig. 4.1; it represents the queue residual busy time at the chunk arrival. The optimal design should lead to have T_{diff} equal to twice the average RTT, so that by sending the offer messages a time equal to $2AvgRTT$ before the last chunk ACK message is received, the bandwidth results continuously utilized and the queue delay minimized[3], i.e., the queue residual time when the selects are returned (T_{queue} in the figure) tends to zero.

If at the chunk arrival the queue residual busy time T_{diff} is too small, say smaller than $2AvgRTT$, the algorithm can foreseen some idle time for the peer; thus, N_a can increase. However, when the peer is slow in distributing chunks or far away from the source, it tends to receive negative selects from neighbors and possibly to stay idle for long time. To avoid flooding the neighbors with an excessive number of useless signaling messages, N_a is increased only if the fraction of positive select messages is larger than the threshold CR.

3.4 Performance Evaluation

3.4.1 Simulation scenario and assumptions

All results shown in this chapter have been obtained through *P2PTV-sim* [18], an open source event driven simulator developed within the NAPA-WINE project. In our scenario, peers are partitioned in four classes based on their upload capacity:

- 15% of peers are in Class 1 with upload bandwidth equal to 5Mb/s ± 10%,

- 35% of peers are in Class 2 with upload bandwidth equal to 1Mb/s ± 10%,

[3]Twice the minimum RTT would be enough to guarantee that a select message is received before the transmission queue empties. However, due to the variability of RTT and the randomness of peer selection process, using the average RTT is a safer choice.

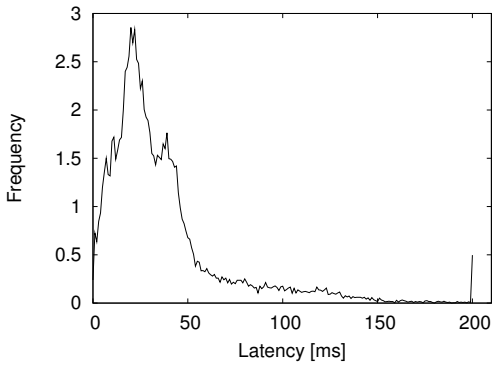

Figure 3.2. Latency distribution taken from Meridian Project [1].

- 30% of peers are in Class 3 with upload bandwidth equal to 0.64Mb/s ± 10%,
- 20% of peers are in Class 4 with negligible upload bandwidth.

The video source belongs to Class 1 peers. The corresponding average bandwidth is $E[B_p] = 1.3$Mb/s. To study the system under several values of network load we change the video rate r_s, so that the load is

$$\rho = r_s/E[B_p].$$

Chunk size is fixed and equal to $L = 100$kb, i.e., about 8 UDP packets (with typical 1500B size), while the inter-chunk time depends on the video rate. In each simulation the source emits 2000 chunks, which are equivalent to a video of about 4min at $r_s = 0.8$Mb/s. D_{max} is set to 7s. We consider $N = 2000$ peers. According to the assumption that the bottleneck is at the peer upload link, the model of the network end-to-end path is almost transparent: it is simply modeled by a delay l_{pq} that is added to the transmission time of all the packets flowing from a to q. End-to-end latencies l_{pq} are taken from the experimental data set of the Meridian project [1]. Latency frequencies are reported in Fig. 3.2, in which values of $l_{pq} \geq 200$ms are accumulated in the last bin for simplicity; the overall mean latency is $E[l_{pq}] = 39$ms.

The overlay topology is randomly generated at the beginning of a simulation by letting each peer randomly select $K = 20$ other peers as its neighbors. Since connections are bidirectional, the average number of neighbors for a peer is equal to $2K$. The topology is static for the whole simulation run (as already mentioned, since we simulate a few minutes of the system behavior, we neglect the effect of churning). All results presented below (except for the time evolution) are obtained averaging the results of four random topologies; when different systems are compared, they use the same four topologies.

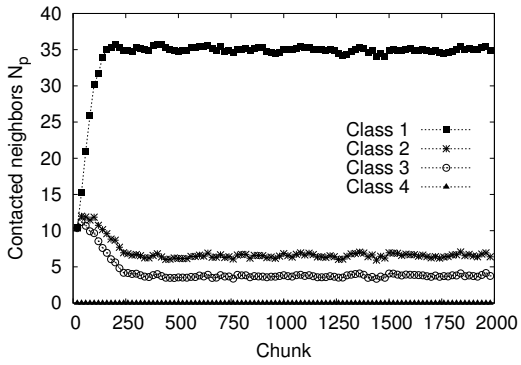

Figure 3.3. N_a evolution versus time with ASP with $CR = 0.0$ and $\rho = 0.9$.

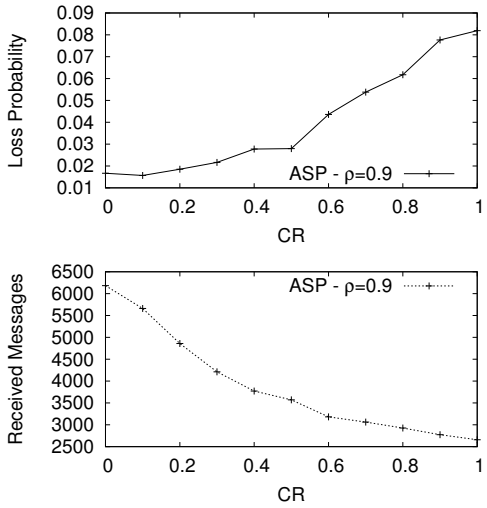

Figure 3.4. Chunk loss probability (top) and total number of signaling messages per peer versus CR (bottom) at $\rho = 0.9$.

3.4.2 ASP transient analysis

We first show the evolution of N_a with time. In Fig. 3.3 ($\rho = 0.9$ and $CR = 0.0$) N_a is averaged over all peers in the same class, considering time windows of 20 chunks (that corresponds to 1.7 s). Starting from the initial $N_p = 10$ for all the peers, the setting of N_a quickly converges (and then remains stable) to the proper value that depends on a upload bandwidth. Clearly, peers with negligible uplink capacity (class 4) do not generate any offer message.

Then, we analyze the impact of the clipping ratio on the performance of the ASP algorithm. We set the load to $\rho = 0.9$, i.e., video rate $r_s = 1.1$Mbps, and we plot the loss probability and the average number of signaling messages sent per peer in Fig. 3.4. The curves show that there is a trade-off between losses and signaling

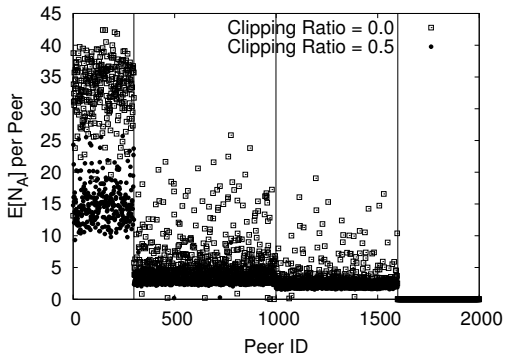

Figure 3.5. Average number of contacted neighbors N_a with ASP algorithms for all the peers, that are in decreasing order with their bandwidth at $\rho = 0.9$.

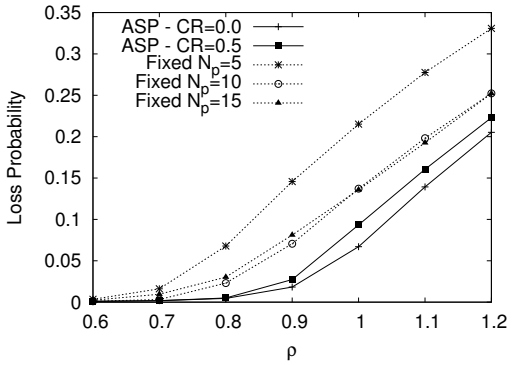

Figure 3.6. Chunk loss probability versus load.

overhead. As CR increases, loss probability increases but the signaling overhead decreases; indeed, due to the epidemic and random chunk diffusion process, number of positive selects decreases by reducing the number of peers that are contacted. To achieve low loss probability, many peers should then be contacted, i.e., many messages should be sent, clearly increasing the signaling overhead. In the following, we will consider two cases: no clipping, $CR = 0$, and $CR = 0.5$, that seems a reasonable trade-off between signaling overhead and loss probability.

Fig. 3.5 reports the average number of contacted neighbors in every offer session ($\rho = 0.9$). Notice that peers are clusterized in the four different classes in decreasing order with their uplink capacity. ASP nicely adapts N_a to the peer upload bandwidth, and the number of contacted peers is roughly proportional to the peer upload bandwidth, The variability of N_a within the class is due to the different position of the peers in the overlay topology: peers that are close to the source tend to have a large number of positive selects from their neighbors and they can effectively exploit their bandwidth by only emitting a limited number of offer messages (N_a is

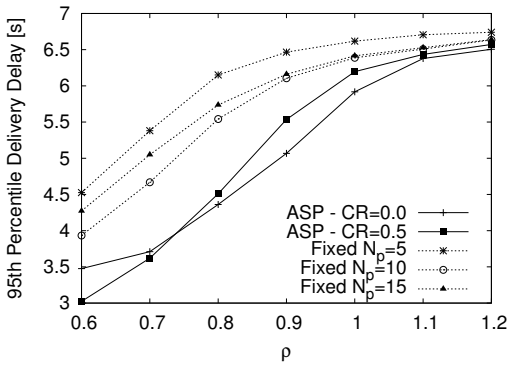

Figure 3.7. 95th percentile of chunks delivery delay versus load.

small); on the contrary, peers that are far away from the source end up emitting a large number of offer messages (N_a is large). Squared markers refer to a scenario in which the clipping ratio, CR, is set to 0.5, while crosses indicate no clipping ratio. The absence of clipping ratio makes peers that are far from the source pointlessly increase N_a to very large values.

3.4.3 Performance analysis and comparison with fixed N_a schemes

We now consider the performance of ASP with respect to schemes in which N_a is fixed, so that every peer always generates the same amount of offer messages, independently on its upload capacity and status of the transmission queue. Fig. 3.6 reports loss probability versus load for the case of ASP with $CR = 0$ or $CR = 0.5$ (solid lines) and N_a fixed to 5, 10, or 15 (dashed lines). Loss probability is averaged over all chunks and all peers. Observe that by guaranteeing a better utilization of the bandwidth, ASP always achieves better performance than the scheme with fixed N_a, e.g., losses are reduced by a factor up to 4 for $\rho > 0.9$. Moreover, improvements are equal to all classes of peers, so that loss probabilities are the same for all classes. Interestingly, ASP outperforms also the system with $N_p = 15$ that corresponds to the value achieved by high bandwidth peers under the ASP (as can be observed by Fig. 3.5). The reason is that $N_p = 15$ is too large a value for low bandwidth peers that end up transmitting lots of chunks, but introducing additional queuing delays to the chunk delivery time. Conversely, when N_a is fixed and equal to 5, peers cannot fully exploit their bandwidth, and this explains the higher loss probability. Same considerations can be achieved from Fig. 3.7 where the 95th percentile of delivery delays of chunks is reported. Again ASP with $CR = 0$ or $CR = 0.5$ show lower delivery delays and, therefore, better performance than schemes with fixed N_a. An interesting fact to point out is that a larger clipping ratio can actually help

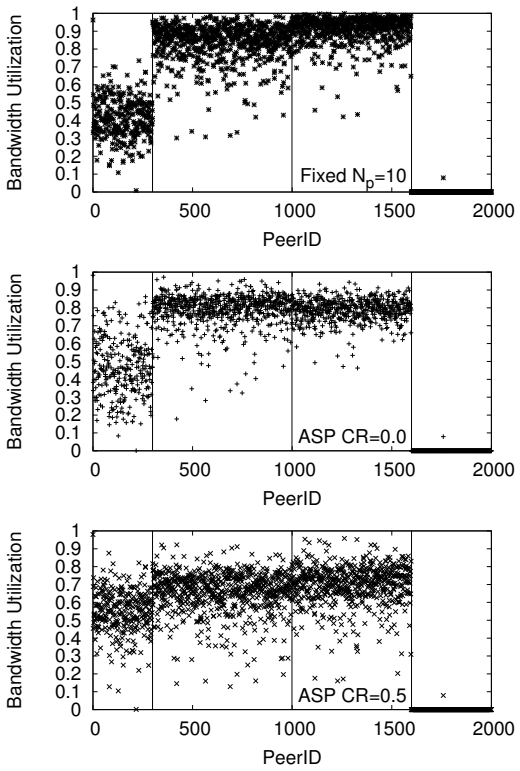

Figure 3.8. Bandwidth utilization with fixed $N_p = 10$ (top), ASP with $CR = 0$ (middle), ASP with $CR = 0.5$ (bottom) at $\rho = 0.6$.

Table 3.1. Average bandwidth utilization for classes 1, 2, 3 and Jain Fairness Index at $\rho = 0.6$.

	Class 1	Class 2	Class 3	Fairness Index
$N_a = 10$	0.412	0.836	0.904	0.934
ASP (CR = 0.0)	0.464	0.792	0.785	0.951
ASP (CR = 0.5)	0.550	0.657	0.695	0.962

in reducing delays when the system is under loaded ($\rho < 0.7$. Indeed the $CR = 0.5$ curve exhibits smaller delivery delays respect to $CR = 0.0$ one if $\rho < 0.75$. This is due to the fairer peer uplink capacity utilization as discussed in the following.

3.4.4 Bandwidth allocation among peers

Fig. 3.8 reports the bandwidth utilization per peer measured as the fraction of time the uplink channel is used to transmit chunks; average values per class and Jain's

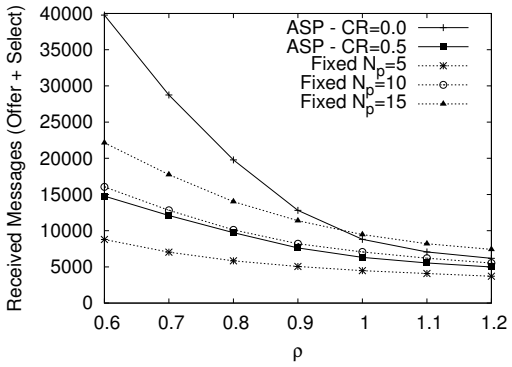

Figure 3.9. Total number of received signaling messages per peer versus load.

fairness index are reported in Tab. 3.1. We consider a scenario in which the video rate is 0.7Mbps, corresponding to $\rho = 0.6$, meaning that each peer can contribute to the chunk distribution by spending only 60% of its upload capacity. Top plot refers to the case of fix $N_p = 10$. High bandwidth peers have a low utilization of about 40% of their bandwidth, meaning that they are basically underutilized due to the low number of contacted neighbors (N_P is low). Class 2 and 3 peers compensate by working most of the time, so that they experience utilization higher than 80%. While this bandwidth allocation still allows to successfully deliver all chunks to all peers (no losses are experienced), the overall delay is quite large, due to the slow store-and-forward at low bandwidth peers.

The ASP scheme is fairer in distributing workload among peers proportionally to their bandwidth, as can be observed by middle and bottom plots, respectively referring to $CR = 0$ and $CR = 0.5$. In this case, the high bandwidth peers automatically increase the number of offer messages, trying to increase the uplink bandwidth utilization. Notice the beneficial impact of the $CR = 0.5$, which, by limiting N_a, reduces the utilization of the low bandwidth peers. This speeds up the chunk delivery time, as already noted in Fig. 3.7.

3.4.5 Signaling overhead

Let us now focus on the signaling overhead. Fig. 3.9 reports the average number of exchanged signaling messages per peer. The signaling overhead decreases with the load due to the larger probability of positive selects per offer message, as confirmed also by Fig. 3.10 which reports the fraction of positive select messages versus all select messages. The case of fixed $N_p = 5$ leads to the smallest number of signaling messages. However, $N_p = 5$ leads also to the worst performance (see Fig. 3.6). Improvements can be achieved for higher values of N_a and for the ASP algorithm. When the load is low, queues are short, chunks are distributed quickly, and peers get

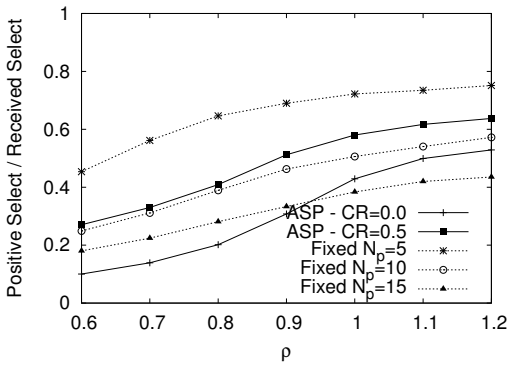

Figure 3.10. Fraction of positive select messages versus load.

them easily, so that most of select messages are negative; when no clipping is used, $CR = 0$, the number of signaling messages is very high and the fraction of positive selects is low. If, instead, clipping is active (ASP with CR =0.5), the mechanism is much more efficient and the waste due to signaling reduces by a factor 3.

Chapter 4

Hose Rate Control

4.1 Introduction

Considering the trading scheme, the most critical parameter is the number of "offers" (messages advertising the list of possessed chunks) a peer should send in parallel, i.e., the number of active *signalling threads*. If this number is too small, the delivery rate of chunks is small, thus upload bandwidth is under-utilized. Conversely, if this number is too large, the committed workload would overflow transmission resources, impairing perceived quality of the video stream.

In this chapter, we show a scheme called *Hose Rate Control*, HRC, to automatically adapt the number of signalling threads to the current network scenario. By doing so, the scheme implements a peer aggregate transmission rate control that aims at jointly exploiting the peer upload capacity and improving QoE of users, reducing as much as possible chunk delivery delays. In other words, the scheme controls the bandwidth allocation on the peer uplink channel.

The HRC objective is to exploit the upload bandwidth of peers while not increasing queuing delay, therefore targeting a less-than-best-effort policy being less aggressive than the TCP congestion control whose regulation scheme is loss-based. This is an explicit design choice that aims at tightly controlling chunk delivery delay and chunk delivery probability, i.e., minimizing packet loss and lengthly retransmissions.

Considering file sharing P2P systems the problem of controlling the sending rate of peers is a timely problem. Recently, BitTorrent designers decided to adopt UDP in their client. Indeed, a new congestion control algorithm called LEDBAT (Low Extra Delay Background Transport) has been proposed [19]. LEDBAT targets a less aggressive congestion control mechanism than the one implemented in TCP.

To this extent HRC is somehow similar to LEDBAT, however, they differ in the following two key aspects: i) HRC is an *aggregate hose mechanism* that controls the

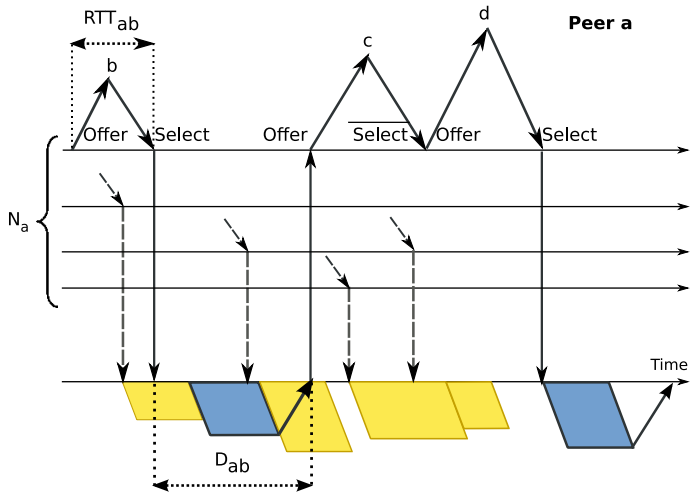

Figure 4.1. Schematic representation of the peer chunk trading mechanism.

overall sending rate of a peer, and not the per-flow end-to-end sending rate; ii) HRC is chunk based rather than packet based. As previously mentioned, this stems from the different goals of streaming versus file sharing P2P applications: the first targets live distribution of un-elastic content, while the latter targets the maximization of download throughput.

We implemented HRC in *PeerStreamer* [14], the P2P-TV application developed within EU-FP7 Network-Aware P2P-TV Application over Wise Networks (NAPA-WINE) STREP project [13]. This allows us to provide experimental results on swarms of up to 1000 peers in a controlled environment. Extensive experimental results obtained considering both simulations and the actual implementation show that, with respect to non adaptive mechanisms, HRC optimizes resource utilization, consistently improving system performance and QoE that we evaluate on real video sequences by computing the SSIM (Structural Similarity Index) [20].

4.2 The Chunk Trading Mechanism

The signalling mechanism used to exchange chunks is a trading scheme similar to the one used in other mesh-based P2P-TV systems [8, 11, 12]. A chunk is sent from a peer to one of its neighbors after a trading phase. Peer a maintains a number of *trading threads*, N_a. Each trading thread evolves as follows:

1) Peer a chooses one of its neighbors b and sends it a signalling message, called *offer* message; it contains the set of younger than D_{max} chunks a possesses.

2) Upon receiving the offer message, b replies with a *select* message to request a desired chunk. Once a chunk has been "selected", the receiver sets it as *pending*

until it is correctly received; a pending chunk cannot be requested and cannot be published yet.

3) When the select message is received by a

- **a)** if a chunk was requested in the select message (*positive select*), a schedules its transmission, inserts it in its chunk *transmission queue* that is served in a FIFO order.
- **b)** Once b has completely received the selected chunk, it sends an ACK message to a.
- **c)** When a receives the ACK message, it can send a new offer message and a new cycle starts.
- **d)** If no chunk was requested in the select message (*negative select*), a can send a new offer message and a new cycle starts.

Peer a is committed to send all the chunks requested in all the received positive select messages. Timers protect the waiting for messages so that in case no reply is received within a timeout, the status is reset and a new thread can start. Fig. 4.1 represents the signalling messages and chunks exchanged by peer a with its neighbors over time. In particular signalling messages/chunks associated to one active thread are highlighted. Note that all N_a trading threads continue these cycles independent of each other.

Several design choices impact the performance of the trading mechanism: 1) the criterion to select destination peers for the offer message – known as the "peer selection"; 2) the strategy according to which peers receiving an offer message select chunks to download – known as the "chunk selection"; 3) the number N_a of threads peer a handles which is somehow equivalent of the window size in a window protocol and represents the rate at which potential transmitting peers offer their chunks.

For the *peer selection* and the *chunk selection* policies we make the simplest possible choices: peer a chooses peers to contact uniformly at random within the set of its neighbors, and the neighbors choose the chunks to select at random among the ones they need. This policy is also known in the literature as "Random Peer - Random Useful Chunk selection" [21].

The key parameter that requires to be set in this mechanism is N_a.

4.3 The Core of Hose Rate Control

HRC, the adaptive signal mechanism that we propose, stems from the basic idea to control the rate at which chunks are sent by peer a by controlling the number of parallel active threads N_a, so that the queuing delay at the transmission queue is at a given (small) target. The rationale is that N_a controls the amount of work that the peer a has to do: if it is too large, upload capacity is exceeded and delay increases, deteriorating performance; if it is too small, the peer available upload bandwidth

could not be well exploited. The rule to decide N_a is based on an estimation of the queuing delay incurred by chunks in the transmission queue: if the queuing delay is large, N_a is decreased, and vice-versa.

More in detail, the algorithm according to which N_a is made adaptive is the following. Let W_a be the internal control variable, which takes real values: $N_a = \max(1, \lfloor W_a \rfloor)$. $W_a < W_{\max}$, where $W_{\max}$ limits the maximum number of offers in flight, e.g., it can be constrained to the number of neighbors of peer a. For every neighbor peer b, peer a maintains an estimate of the minimum Round Trip Time. This estimate can be computed/updated by a every time it receives a select message as the difference between the time the select message is received and the one the offer is sent,

$$RTT_{ab} = t^{(a)}_{\text{rx,select}} - t^{(a)}_{\text{tx,offer}}$$

where $t^{(p)}_{\text{action,type}}$ identifies the time of the "action" triggered by the message of "type" at peer p; action=$\{rx,tx\}$, type=$\{offer,select,chunk,ack\}$.

When a receives an acknowledge from b, it estimates the delay D incurred by the chunk in the transmission queue, as $\hat{D} = t^{(a)}_{\text{rx,ack}} - t^{(a)}_{\text{rx,select}} - RTT_{ab}$, i.e., subtracting a RTT from the difference between the time at which the acknowledge was received and the time at which the chunk was enqueued. $\hat{D}$ is then compared with a prefixed target value, D_0, and W_a is updated according to the following rule:

$$W_a(n) \leftarrow W_a(n-1) - K(\hat{D} - D_0) \tag{4.1}$$

N_a is then increased/decreased by $\Delta N_a = \lfloor W_a(n) \rfloor - \lfloor W_a(n-1) \rfloor$. Now, if $\Delta N_a = 0$, the number of active threads is not changed, and peer a is allowed to send a new offer to one of its neighbors. If $\Delta N_a > 0$, the number of active threads is increased, and peer a is allowed to send two or more offers to its neighbors. At last, if $\Delta N_a < 0$, the number of active threads is decreased and the current thread is stopped (no new offer is sent).

Note that targeting queuing delay, HRC results less aggressive than TCP congestion control which reacts to congestion only after a packet has been dropped by the queue. This is an intended behavior of HRC since having a tight control of chunk delivery delay is fundamental to make the system to work properly in P2P-TV streaming applications. On this regards, observe that P2P streaming applications can effectively exploit spatial diversity (every peer can retrieve a chunk from several neighbors) to effectively face congestion that may arise at some of the peers.

4.4 Performance Evaluation by Simulation

4.4.1 Simulation scenario and assumptions

All simulation results shown in this document have been obtained with *P2PTV-sim* , an open source event driven simulator available at [18]. In our scenario, peers are partitioned in four classes based on their upload capacity: 15% of peers are in Class 1 with upload bandwidth equal to 5Mb/s ± 20%, 35% in Class 2 with 1.6Mb/s ± 20%, 35% in Class 3 with 0.64Mb/s ± 20%, 15% in Class 4 with negligible upload bandwidth. The video source belongs to Class 1. The average bandwidth per peer is $E[B_a] = 1.25$Mb/s.

In each simulation D_{max} is set to 6s if not otherwise stated. We consider $N = 2000$ peers. According to the assumption that the bottleneck is at the peer upload link, the model of the network end-to-end path is almost transparent: it is simply modeled by a delay l_{ab} that is added to the transmission time of all the packets flowing from a to b. End-to-end latencies are taken from the experimental dataset of the Meridian project [1]; the overall mean latency is $E[l_{ab}] = 39$ms.

The well-known *Pink of the Aerosmith* video sequence, encoded using the H.264/AVC Codec, is considered as benchmark. A hierarchical type-B frames prediction scheme has been used, obtaining 4 different kinds of frames that, in order of importance, are: IDR, P, B and b. The group of pictures (GOP) structure has been set to IDR x 8 {P,B,b,b}. The video consists of 3000 frames, which correspond to about 120s of visualization. The nominal video rate of the encoder r_s is a free parameter that we vary to enforce different values of the system load defined as,

$$\rho = r_s / E[B_a] \tag{4.2}$$

The source node generates a new chunk at regular time, i.e., every new frame. 40B long signalling messages are considered.

The overlay topology is randomly generated at the beginning of a simulation by letting each peer selects 30 other peers at random as its neighbors. Since connections are bidirectional, the average number of neighbors for a peer is approximately equal to 60. As we simulate a couple of minutes of the system behavior, we neglect the effect of churning so that the topology is static for the whole simulation run. All plots presented below (except for the time evolution) are obtained averaging the results of four random topologies; when different systems are compared, they use the same four topologies.

Real video streams carry highly structured information, part of which is more important than other, with high variability in the generated bit-rate. Chunk loss probability and delivery delay are the performance indexes typically adopted by the networking community, but they provide only a partial view of the actual performance of a P2P-TV system, the user perceived quality. In the multimedia and signal

processing communities, instead, the evaluation of the perceived quality is considered mandatory, see [22, 23] for notable examples. To this extent, performance is expressed in terms of average Structural Similarity Index (SSIM) [20] which has been designed to improve on traditional methods like Peak Signal-to-Noise Ratio (PSNR) and Mean Squared Error (MSE), which have proved to be inconsistent with human eye perception. The SSIM is a measure of the similarity of the received image compared against the original source. It is a highly non linear metric in decimal values between -1 and 1. Values above 0.95 are typically considered of good quality. In our simulation scenario, SSIM has been computed considering video frames received by 100 peers (25 for each class), and then averaging among all of them. The initial and final 10s of video have been discarded to focus on steady state performance.

4.4.2 Transient analysis

In the following, we show results about the HRC algorithm obtained through simulations; D_0 is set to 100ms. The source has rate $r_s = 1.2$Mb/s, corresponding to $\rho = 0.95$.

• **Simple scenario** - Let us focus on a randomly selected peer a with available upload bandwidth of 4Mb/s that varies due to interfering traffic. Fig. 4.2 reports the evolution over time of queue delay $\hat{D}(t)$ (top), the number of active threads N_a (center) and the throughput (bottom) for peer a. During the first 20s, no interfering traffic is present; after an initial transient the value of N_a stabilizes around 25, so that the peer can exploit at best its upload bandwidth. At time $t = 20$s, a Constant Bit Rate flow starts injecting 3Mb/s of interfering traffic in the uplink. $\hat{D}(t)$ abruptly grows; thus, N_a is reduced and stabilized around 5 and the upload throughput decreases to 1Mb/s. At $t = 80$s the whole upload bandwidth turns to be available again, inducing an increase of N_a which gets back to 25. Then, from $t = 100$s to $t = 120$s, a TCP-like flow is present, consuming the whole peer's upload bandwidth. In this period, the number of active threads drops to its minimal possible value, $N_a = 1$, because the congestion due to the TCP-like flow pushes $\hat{D}(t)$ over the control target D_0. As a consequence, the application throughput is reduced to negligible values. In conclusion, the control algorithm succeeds in promptly reacting to bandwidth variations, and in achieving less-than-best-effort bandwidth utilization.

• **Flash crowd scenario** - We now consider the scenario in which the system operating point is abruptly modified at $t = 30$s: a sudden ingress of 400 new peers with negligible upload-bandwidth and a contemporary reduction by 50% of the available upload bandwidth of all peers belonging to Class 3 happens. Given video rate $r_s = 1$Mb/s, this causes the system load ρ to shift from 0.8 to 1.1. Even if this scenario is rather artificial, it has been selected because it maximizes the stress on the control scheme. Fig. 4.3 reports the evolution of N_a (center) and throughput

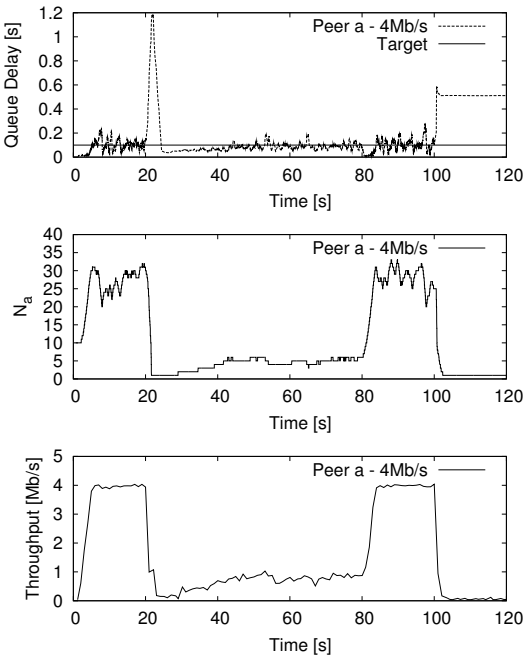

Figure 4.2. Queuing delay (top), value of N_a (center) and throughput (bottom) vs. time with variations due to interfering traffic on upload link. $\rho = 0.95$

.

(bottom) for three sample peers, a, b and c, with upload bandwidth of 4, 2 and 1Mb/s, respectively. The evolution of peer a queue delay $\hat{D}(t)$ is also reported (top). When $\rho = 0.8$, N_a, N_b and N_c adapt to different values, reflecting each peer's ability to contribute to chunk diffusion. Since $\rho < 1$, not all system capacity is required, and N_a rapidly grows to its maximum value $N_a = W_{\max} = 53$, i.e., the number of a neighbors. At $t = 30s$, the HRC system reacts to the sudden system condition variations. In particular, for the high bandwidth peer a, N_a initially increases, since the number of its neighbors increases and its capacity was not fully exploited (its queuing delay still being smaller than D_0). Then, the increased system load boosts the percentage of offers that are positively selected, causing additional queuing delay, so that N_a decreases. After a quick transient, upload rate matches each peer upload capacity, and queuing delay reaches the target D_0.

4.4.3 Steady-state analysis

In this section, we focus on the steady-state performance of HRC and we compare it with non-adaptive schemes that use a fixed value of N_a.

Fig. 4.4 compares the HRC system for $D_0 = 150$ms and 200ms and the non-adaptive schemes in which N_a is fixed. The video rate r_s is increased (reported on

bottom x-axis) to observe the performance of the system of increasing ρ (reported on top x-axis). When $\rho < 1$, the SSIM increases for increasing r_s thanks to the higher quality of the encoded video (Encoded Video Quality, EVQ, curve in the plot). As soon as the system is overloaded, the SSIM rapidly drops due to missing chunks which impair the quality of the received video. In all scenarios, HRC outperforms the non-adaptive scheme, for any values of N_a. Schemes with too small values of N_a do not fully exploit the system bandwidth, e.g., $N_a = 10$; schemes with too large values of N_a tend to overload the peer transmission queue leading to an unnecessary increase of the chunk delivery delay, e.g., $N_a = 40$. The performance of the scheme with $N_a = 20$ are comparable with that of HRC. However, setting the value of N_a is very critical, since the optimal value depends on many other system parameters, such as the peers upload bandwidth distribution, that, besides being difficult to know, are variable in time due to interfering traffic, as seen in Figs. 4.2 and 4.3.

Fig. 4.5 reports the average SSIM obtained with HRC versus D_0 for different values of the playout delay D_{max}. $\rho = 0.95$. On the one hand, small values of D_0 lead to inefficient exploitation of peer upload capacity. Being the system load quite large, this leads to loss of chunks that impairs the video quality. On the other hand, D_0 represents the average queue delay that a chunk suffers at every hop it traverses; larger values of D_0 lead to larger delivery delays that might translate into

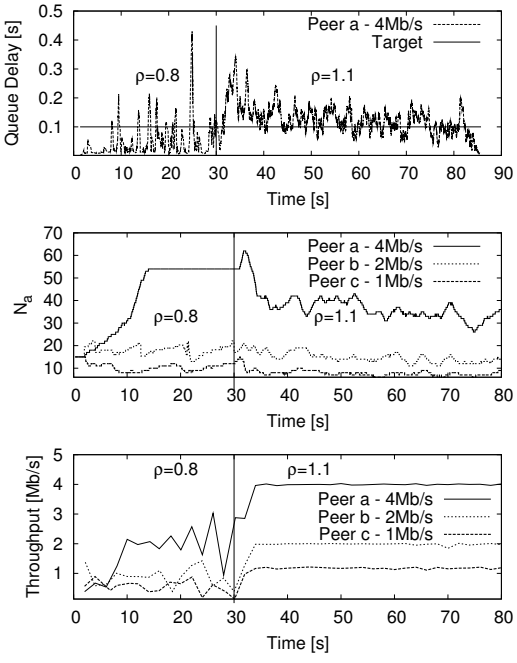

Figure 4.3. Queuing delay (top), N_a (center) and throughput (bottom) vs. time for flash crowd.

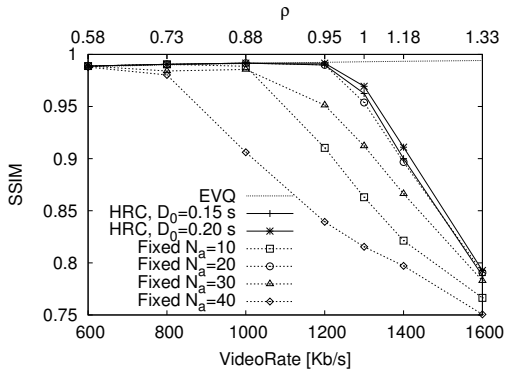

Figure 4.4. Average SSIM of HRC and non-adaptive schemes versus the system load. Simulation results, 2000 peers.

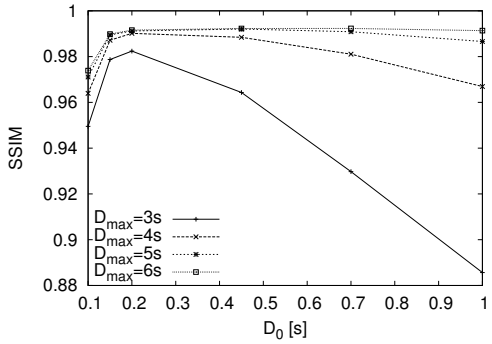

Figure 4.5. Average SSIM for different values of D_0 and D_{max}. $\rho = 0.95$. Simulation results, 2000 peers.

chunk losses for small values of D_{max}, again impairing the QoE represented by SSIM index. Thus, smaller values of D_0 should be preferred. A good tradeoff is obtained for $D_0 \in$ [150,200]ms.

We have performed a more extensive set of simulations to assess the benefits of HRC. Due to lack of space we do not report them here, but we prefer to present some experimental results we collected from real implementation of HRC.

4.5 Performance Evaluation by Experiment

The HRC controller has been implemented into *PeerStreamer* [14] P2P-TV application. In the following we briefly discuss the key aspects of the implementation and provide some experimental evidence of the benefits of HRC.

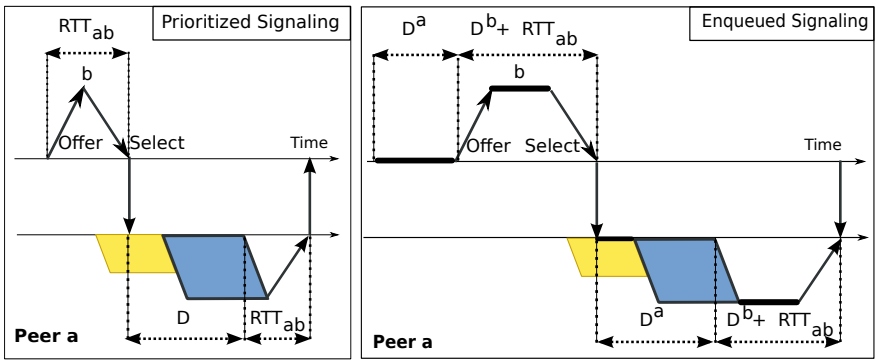

Figure 4.6. Schematic representation of the peer chunk trading mechanism with prioritized signalling -PS- (left) and enqueued signaling -ES- (right).

4.5.1 Implementation issues

The most critical part when undergoing the actual engineering of the HRC scheme is the estimation process of queuing delay which is at the core of HRC scheme. Three different cases can be considered:
i) let us first consider the scenario in which the access device of the bottleneck node supports separate queues: a high priority queue serves signalling packets, and a low priority queue serves data packets. The estimation of the round-trip time between a and b, RTT_{ab} (which does not include the queuing delay), could be easily carried out by exploiting the higher priority service offered to signalling packets: $RTT_{ab} = t^{(a)}_{\text{rx,select}} - t^{(a)}_{\text{tx,offer}}$;
ii) in the second scenario a single class of service is offered by the network devices, but peers are synchronized. Here, signalling messages are delayed by the transmission queue too, as sketched in left part of Fig. 4.6, but synchronization allows to measure the One-Way-Delay between a and b, OWD_{ab}, as the minimum of all $t^{(a)}_{\text{rx,select}} - t^{(b)}_{\text{tx,select}}$ estimates. Even some coarse synchronization, as the one provided by the NTP protocol, would suffice and errors (in the order of 1ms) would only marginally affect the HRC control, since its target queuing delay D_0 is of the order of 100ms;
iii) in the third scenario peers are not synchronized and no priority policy is provided. $\hat{D}$ cannot be estimated anymore, since any RTT measurement includes both the queuing delay at peer a, denote it by $D^{(a)}$, and the queuing delay at b, $D^{(b)}$; i.e., it is only possible to estimate the sum of the queuing delays,

$$\hat{D}_{(a+b)} = D^{(a)} + D^{(b)} = t^{(a)}_{\text{rx,ack}} - t^{(a)}_{\text{rx,select}} - RTT_{ab} \tag{4.3}$$

RTT_{ab} can still be estimated as the minimum over all RTT samples, while it is impossible to decouple $D_{(a+b)}$ from $D^{(a)}$ and $D^{(b)}$. Thus, the HRC algorithm at peer

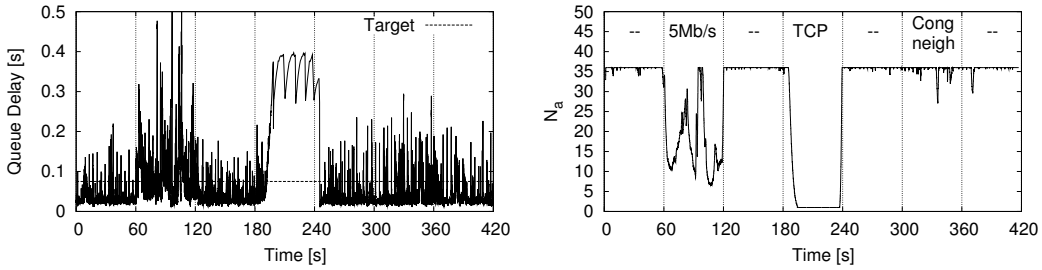

Figure 4.7. Queue delay (left) and N_a (right).

a controls the sum of the queuing delays, and it is coupled with the HRC control of all its neighbors.

We consider this latter scenario in our implementation so that the queuing delay is estimated as in (4.3). At last we emphasize that the implementation of HRC requires to make the system robust to losses of signalling messages and chunks through the use of opportune timeouts (set to 1.5s in current implementation).

4.5.2 Experimental results

• **Simple scenario** - We first consider a simple scenario in which the source s is connected to a HRC-enabled peer a only. 36 other peers are then attached to a, so that its upload capacity is used to feed all neighbors. We then impose transient conditions to the upload link of a: the Linux `tc` tool is used to limit the upload capacity and delay, while the `iperf` tool is used to inject artificial traffic. Video rate is 0.6Mb/s, 20 ms RTT is imposed on all links and $D_0 = 75$ms.

Fig. 4.7 reports the evolution of queuing delay (left) and of N_a (right) for peer a. During the first 60s, peer a uplink bandwidth (100Mb/s) is large enough to transmit all committed chunks. Since a queuing delay cannot reach the target, N_a stabilizes at the neighborhood size ($N_a = W_{\max} = 36$).

Decrease of available capacity - At time $t = 60$s, a uplink capacity is limited to 5Mb/s, inducing HRC to reduce the number of parallel signalling threads while queuing delay varies around the target value. From $t = 120$s to $t = 180$s, N_a stabilizes at the maximum allowed value, being uplink bottleneck removed.

Competing TCP traffic - At time $t = 180$s a competing TCP flow starts consuming the link capacity, increasing the queuing delay so that N_a reduces to 1, the minimum possible value. At the end of the TCP flow, HRC controls N_a value increasing it to 36 again.

Congestion at neighbor - From $t = 300s$ to $t = 360s$, peer b, one of a's neighbors, suffer congestion in its uplink: a TCP flow starts sending data from b to a, so that $D^{(b)}$ grows, possibly impairing $\hat{D}_{(a+b)}$. The plot shows that the estimated queuing

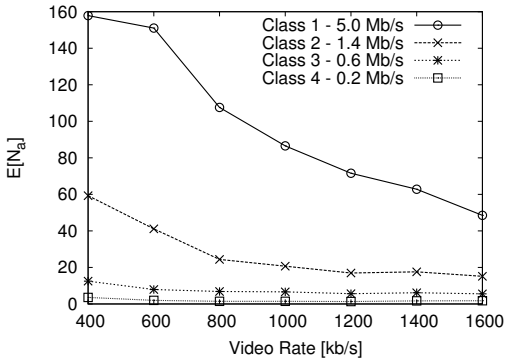

Figure 4.8. Average N_a for HRC when varying video rate r_s. Experimental results in swarm of 1000 peers.

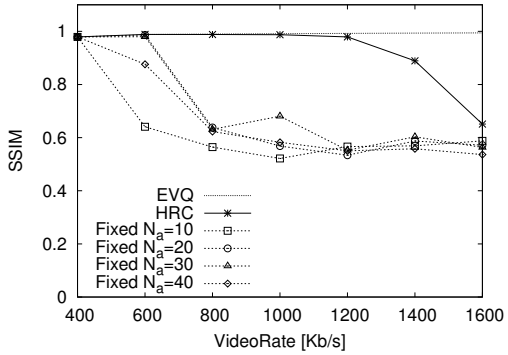

Figure 4.9. Average SSIM for HRC and non-adaptive schemes when varying video rate r_s. Experimental results in swarm of 1000 peers.

delay at peer a is slightly affected by the presence of congestion on its neighbor, indeed some larger oscillations are visible. However N_a is basically unaffected. The intuition indeed suggests that if the number of "biased" estimates at peer a is limited, the system is still able to control the peer uplink queue by "filtering" out the few overestimated samples.

• **Large swarm** - We now present results collected by running the application in a controlled test-bed composed of 200 PCs. Each PC runs 5 copies of the application simultaneously, creating a swarm of 1000 peers. Also in this case, the average number of neighbors for each peer was approximately equal to 60. Each peer upload capacity has been artificially limited using a rate limiter embedded in our P2P-TV application which runs at packet level: 10% of peers have 5Mb/s, 35% have 1.6Mb/s, 35% have 0.64Mb/s and 20% have 0.20Mb/s, corresponding to an average per peer data link capacity of 1.32Mb/s. Latencies among peers randomly varies between 10ms and 20ms (so that the RTT varies in [20,40]ms). The *Pink of the*

Aerosmith video (352x240p resolution, 25fps, H.264/AVC Codec) has been encoded at different rates and "streamed" over the swarm looping the video 5 times. After discarding the initial 12min of video, each peers saves 100s of the received frames on disk. SSIM is then computed against the original YUV video for all video traces; then average SSIM is computed over all peers. Simple random overlay topology and random peer/random chunks selection are adopted. The playout delay D_{max} is set to 6s, the HRC queuing target D_0 is set to 200ms, and the maximum number of offer threads $W_{\max}$ is set to twice the number of current neighbors.

Fig. 4.8 reports the average number of active signalling threads N_a for each class of peers when HRC is enabled and video rate r_s is increased. The first evident thing is that HRC achieves the objective of adapting N_a to actual peer's upload bandwidth: e.g., for video rate 1.0Mb/s, $E[N_a]$ is 90 for high bandwidth peers, $E[N_a]$ is 22 for mid bandwidth peers, $E[N_a]$ is 5 for the low bandwidth peers and $E[N_a]$ is 1 for those peers with very low bandwidth. The second thing to notice is how N_a decreases when r_s increases and, therefore, the load ρ of the system grows; for higher loads, e.g., $r_s = 1.6$Mb/s, the chunks acceptance probability becomes higher inducing peers' uplink queue to grow so that N_a decreases. Instead, when system conditions are relaxed, e.g., $r_s = 0.6$Mb/s, queues are short and peers - especially those whose upload bandwidth is large - increase the number of active trading threads to raise their transmission workload. Note that no fixed values would be suitable for any scenario.

Focusing on the Quality of Experience, again expressed with SSIM index [1], Fig. 4.9 compares HRC behavior against non adaptive schemes in which N_a = 10,30,20,40 respectively. Results are similar to the one of Fig. 4.4: all schemes perform similarly when the system is under-loaded, e.g, $r_s = 400$kb/s, but as soon as r_s increases, HRC dramatically outperforms any fixed schemes. Indeed, the correct choice of N_a is critical: it must be small to prevent from overloading low bandwidth peers, while it must be large to avoid under-utilizing high bandwidth peers. Any fixed values would cause a mismatch, impairing the overall system performance. This is the fundamental concept behind HRC algorithm and it is even clearer when combining results in Fig. 4.9 with those shown in Fig. 4.8: by simply adapting the transmission window represented by N_a to peer's upload bandwidth, performances are greatly improved.

In Fig. 4.10 we report SSIM index for each peer for video rate of 0.8Mb/s (left plot) and 1.6Mb/s (right plot), respectively. The former corresponds to a case in which the system is under-loaded, the latter refers to a scenario in which system conditions are heavily stressed. Again HRC shows much better performance: if system load is low ($r_s = 0.8$Mb/s) HRC guarantees each peer to successfully play the whole video. On the contrary, any fixed value of N_a fails at successfully delivering

[1] SSIM is smaller than 1 since we are considering the encoding loss too.

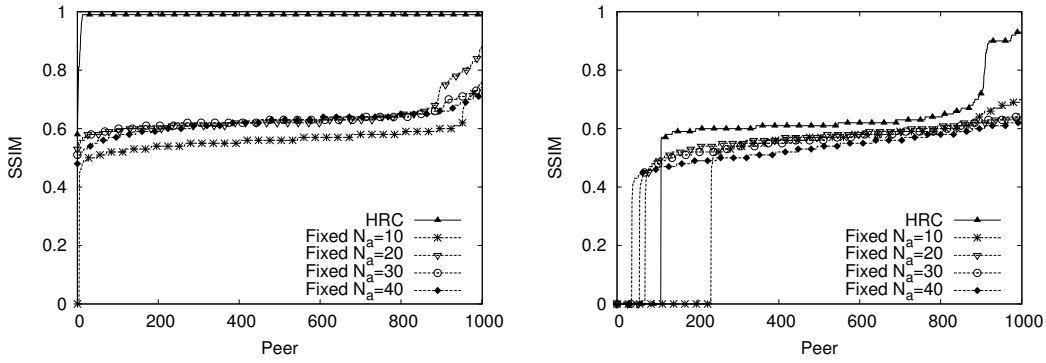

Figure 4.10. Average SSIM per peer distribution for HRC and non-adaptive schemes for video rates $r_s = 800$kb/s (left) and $r_s = 1600$kb/s (right). Experimental results in swarm of 1000 peers.

the content, severely impairing the video quality; finally, when $r_s = 1.6$Mb/s the system has not enough resources to satisfy all peers demand, but, still, HRC shows better performance respect to whatever fixed scheme by efficiently allocating peers' upload capacity.

Chapter 5

Conclusions

In this first part of the document, we focused on the trading phase of pull-based P2P-TV systems. While a large amount of messages exchanged during the trading phase causes transmission queues to grow, increasing chunks delivery delays, losses and wasted time and resources, a strong limitation to the number of these signaling messages leads to bad performance in terms of losses.

To find the best trade-off, we proposed two distributed algorithms called Adaptive Signalling Protocol (ASP) and Hose Rate Control (HRC). They both aim at determining the best rate at which peers should send buffermaps to their neighbors. Their final objective is to efficiently exploit the peer available upload bandwidth by controlling the queuing delay suffered by transmitted chunks in the peer uplink, which is today the typical bottleneck for P2P-TV systems. Our first proposal, ASP, represents a preliminary scheme based on periodic advertisement of peer's buffermap, offered to a subset of peer's neighbors. However, the resulting mechanism assumes scenarios in which latencies among peers are expected to be almost homogeneous, but this is a difficult condition to meet in practice. Indeed, in heterogeneous scenarios implementing the estimation process of T_{diff} represents a hard task. For this reason, we selected only HRC to be implemented in a real P2P-TV client: indeed, HRC proves to be much simpler to implement and flexible with respect to the network scenario variability.

Our results prove that the proposed algorithms actually reduce the amount of signaling overhead, they automatically induce peers to share available resources, introducing a fair and efficient upload bandwidth utilization in different scenarios. Furthermore, ASP and HRC improve system performance in terms of delays, chunk loss probability and, thus, QoE perceived by users.

5.1 Future Directions

Given its implementation in PeerStreamer and the promising results achieved in a controlled environment, HRC is ready to be tested over the real Internet. We are focusing on the evaluation of its effectiveness in a large world-wide distributed environment such as PlanetLab. Plus, we would like to compare its delay-based core mechanism with a more aggressive loss-based one and test its benefits in a scenario in competition with TCP connections.

Part II

Overlay Topology Design

Chapter 6

Introduction

The goal of this part of the document is to provide simple guidelines for the design of new generation network-aware P2P applications that optimize the chunk distribution effectively exploiting some knowledge on the underlay transport network status. Information about the network status is collected through simple end-to-end measurements performed by a distributed monitoring platform embedded within the P2P-TV application. This information can be used locally by peers to guide the neighbors selection process, thus influencing the final construction of the overlay topology at application level (Figure 6.1).

More in detail we show how the logic adopted to create and maintain the overlay topology can leverage on peers upload bandwidth and peer-to-peer distances (in terms of latencies and hop-count) knowledge to achieve the twofold objective of localizing the traffic generated by the P2P-TV application while improving the user experience. We present a rich collection of both simulation and experimental results in support of our considerations. All the experiments presented in this part were performed using PeerStreamer, an Open Source P2P-TV application available at [14] that has been developed within the FP7-ICT NAPA-WINE project [13].

In particular this part aims at shedding light to the following open issues about unstructured P2P-TV systems:

- What is the real impact of latencies and peer access bandwidth heterogeneity on the performance of a P2P-TV system?
- Is it possible to implement smart algorithms that explicitly exploits the knowledge of network metrics and how can the application exploit them?
- How can the application get the required information?
- Which are the possible gains for the application?

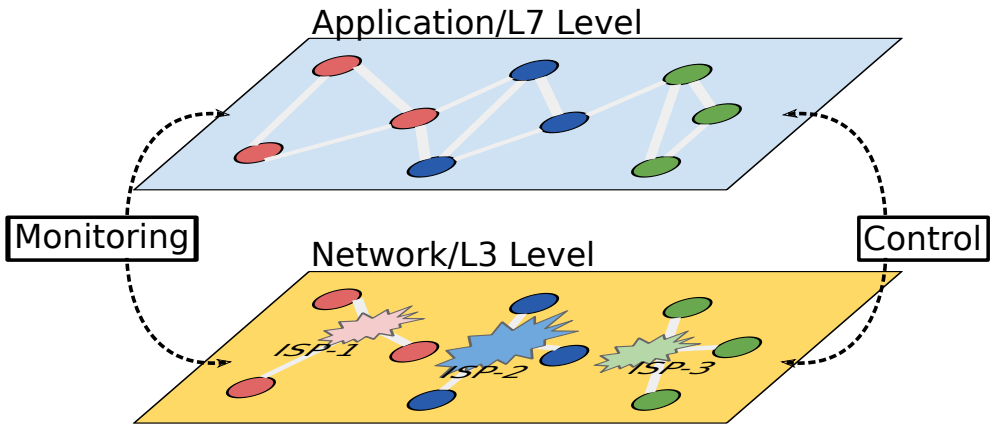

Figure 6.1. Representation of the interaction between application and network levels for a network-aware overlay topology construction.

- How can we reduce then the underlying transport network load by designing a smart system?

We emphasize that building new P2P applications that are network-aware and reduce the underlay transport network resource consumption is an important issue. This problem has received a lot of attention considering P2P file sharing applications (see [24] and references therein), but the same problem is even more important for P2P-TV applications, and the only works on the subject that we are aware of are [25–27]. Therefore, it is still a matter of debate how to design P2P-TV applications that minimize the impact on underlying network, while still guaranteeing good performance for the users.

In Chapter 7 we present a preliminary set of results to show that network-awareness in P2P-TV systems is actually achievable and exploitable. In particular, we want to prove that localization of traffic generated by P2P-TV applications can actually lighten the load on the underlying network, while improving – or at least not impairing – the QoE perceived by users. This conclusion is validated through simulations and test-bed experiments.

Chapter 8 presents a novel algorithm for neighbors selection whose final objective is bringing some smartness into the overlay construction process. Specifically, we show purely experimental results involving more than 1000 peers and we confirm the early results achieved in Chapter 7. Results, obtained comparing 12 different strategies for neighbors selection, prove that we can get a good degree of localization of traffic generated by P2P-TV application while actually improving system performance and the user experience.

Chapter 7

Network Awareness in P2P Streaming Applications

7.1 Introduction

A preliminary version of this chapter has been presented in [28] where the performance of P2P streaming systems with bandwidth heterogeneity, network latencies and realistic properties of encoded video have been analyzed in theory and via simulations. This chapter adds some new content:

1. a working implementation of the P2P-TV system;
2. a set of real in-network P2P measurements supporting the decisions of the network-aware application;
3. measurements results obtained with the real application in controlled networking scenarios;
4. performance and measures on Internet-Wide experiments mixing PlanetLab nodes and peers running on our University campuses.

7.2 Related work

The goal of this Section is not writing a short survey on P2P streaming, but give the correct perspective to the contribution of this part of the document. We restrict our the analysis to papers dealing with the control/optimization of the chunk distribution process in P2P-TV systems, with specific attention to works with implemented systems based on real measures.

Two key features to optimize the chunk distribution process in unstructured P2P-TV system are: the *chunk scheduling algorithm* according to which peers exchange chunks, and the logic adopted to create and maintain the *overlay topology*. For the *chunk scheduling*, several algorithms have been proposed and analyzed in an

idealized scenario in which each peer is supposed to instantaneously have a perfect view of the internal state of other peers and, in particular, of the chunks they need [5–7, 15, 29]. This assumption, however appears rather far from reality in light of the fact that in practical settings the typical end-to-end latencies between peers are usually comparable or even larger than time constants associated to chunks dynamics. The impact of latency on system performance has been analyzed by means of a simple model corroborated by real measurements in PlanetLab in [8]. The authors propose a system that mitigates the effect of latency introduced by the need for exchanging state information. Furthermore, while many of the previous works have considered homogeneous scenarios in which all peers are indistinguishable, some papers specifically focus on the impact of peers bandwidth heterogeneity and how it can be exploited to improve system performance [9, 15, 30]. Delay (instead of bandwidth awareness) is considered in [31] for scheduling chunk transmission on a randomly built overlay. These aspects seem fundamental in building a real system, peers homogeneity being hardly met in practice where narrow bandwidth residential users coexist with larger bandwidth business/residential users. More recently, an experimental comparison of different software streaming approaches has been reported in [32]. This chapter, combined with Chapter 8, is somehow complementary this latter, since it focuses on chunk scheduling algorithms and their performance, assuming the overlay topology to be a full mesh.

The *overlay topology* design has been investigated less deeply and the system performance has been typically analyzed assuming the overlay topology to be either a fully connected mesh or an homogeneous random graph. In [33] the problem of building an efficient overlay topology, taking into account both latency and bandwidth, has been formulated as an optimization problem; however, the complex interactions between overlay topology structure and the chunk distribution process are completely ignored in [33], where continuous streams of information are distributed (in a purely push fashion) among peers.

In [16] a theoretical investigation of optimal topologies is formulated, considering latency and peer bandwidth heterogeneity; scaling laws are thus discussed. In [17], a distributed and adaptive algorithm for the optimization of the overlay topology in heterogeneous environments has been proposed, but network latencies are still ignored. The authors of [34] propose a mechanism to build a tree structure on which information is pushed. They combine two ideas: good topological properties are guaranteed by means of prefixes based on peers identifiers (similarly to what is done in other structured P2P systems) and latency awareness is used to select a specific peer between those with the same prefix. Similar in spirit, but in unstructured systems, we propose in this chapter an overlay topology design strategy that, taking into account latency and peer heterogeneity, aims at creating an overlay with good properties and low chunk delivery delays. In highly idealized scenarios, [35, 36] show with simple stochastic/fluid models that overlay topologies with small-world

$\mathcal{N}$	Set of peers; $\|\mathcal{N}\| = N$
$\mathcal{K}_p$	Set of p selected neighbors; $\|\mathcal{K}_p\| = K_p$
$\mathcal{D}_p$	Set of neighbors desirable by p;
B_p	Peer p upload capacity [bit/s]
l_{pq}	RTT between peer p and q [s]
h_{pq}	Number of IP hops between peer p and q
D_{max}	Playout delay [s]
r_s	Average video rate [bit/s]
L	Average chunk size [bit]
α	Probability of selecting a peer at random
T_{bw}	Bandwidth preference threshold [bit/s]
T_{RTT}	Latency preference threshold [s]
T_{hop}	Hops preference threshold
δ	Topology scaling factor - relates K_p to B_p
β	Signaling scaling factor - relates N_p to B_p
λ_p	Offer rate of peer p [1/s]

Table 7.1. Notation and formalism adopted.

properties can be effectively exploited to support chunk distribution in P2P-TV systems. In some sense the findings in [35, 36] provide a theoretical foundation to our strategies, but the scenarios considered there are far from being realistic.

At last it is worth to mention [37], where the authors experimentally compare the performance of unstructured systems and structured, multiple-tree based systems. Results in [37], indicates that unstructured systems tends to outperform tree-based systems in highly dynamic scenarios as well as in scenarios characterized by bandwidth limitations.

This chapter shows that in P2P-TV systems localizing traffic actually improves network friendliness like shown in [25], but focusing also on users' side, we prove that not only quality of experience of users is not reduced due to localization, but it is even improved.

7.3 System Description

As already seen before in this document, we consider a system in which a single source peer encodes and seeds the content into the P2P network. The video stream is chopped into "chunks" according to some policy, and chunks are exchanged among peers without a predefined structure. Table 7.1 summarizes the notation we use throughout this chapter. Let $\mathcal{N}$ be the set of peers, with cardinality N. Since the application satisfies near real-time constraints, every generated chunk must be

received within a deadline from the moment it is emitted by the source; let this deadline be the *playout delay*, D_{max}. After the playout delay expires, the chunk is not traded anymore. Our goal is the design of a configurable, network-aware application that exploits peer-based measurements of network state, e.g., bandwidth, delay and number of IP-level hops, to reduce the overall network load while possibly increasing the high end-user quality. RTT and number of hops are easy to measure; the upload bandwidth can be estimated with some collaborative approach as recently proposed in [38].

Chunks are exchanged among peers that are *neighbors* to each other. The peer neighborhoods determine the structure of the overlay topology, which can be represented as an undirected graph $G(V,E)$, where $(p,q) \in E$ if and only if $p \in \mathcal{N}$ and $q \in \mathcal{N}$ are neighbors to each other. The overlay topology evolves dynamically, but its dynamics are slower than chunk distribution dynamics (seconds to minutes versus tens/hundreds of ms). Overlay topology changes are driven mainly by peer churning and by the algorithm that peers implement to dynamically select neighbors. Central authorities that support the overlay maintenance as the one proposed by the IETF [39] are compatible with this scenario, but not essential. We envision a fully distributed mechanism in which peers can actively measure network characteristics and then evolve their neighborhood based on those as in [17, 40].

Turning our attention to the chunk trading mechanism, we propose an offer/select protocol, where peers explicitly signal to one neighbor the availability of chunks through offer messages; the neighbor explicitly selects the chunk to be transferred. This two-way handshake is required to avoid peers to receive multiple copies of the same chunk as already discussed in Part I.

Again, we assume the main system bottleneck to be constituted by the upload bandwidth of peers. Indeed, in today Internet the popularity of ADSL access technology offers end-users a large amount of download capacity, but a much limited upload bandwidth.

7.4 Design Choices

Given a system as described in Sect. 7.3 there are many design options open to define the strategy to build. These are discussed in the following sections.

7.4.1 Overlay topology design

The overlay topology design should take into account three different aspects: *i)* peers with small RTTs should be connected with each other to reduce the offer/select signaling delay; *ii)* contrasting the goal above, random choices help building robust graphs, limiting the risk of building highly clustered topologies with poor diameter

properties; *iii)* as already shown in [15, 36], high upload bandwidth peers should be highly connected to each other and to the source, so that they can effectively contribute to the chunk distribution. Based on these principles, we consider RTT, IP-hop count and upload capacity as key measurements to define the overlay topology formation strategy.

In particular, a peer p selects a subset $\mathcal{K}_p$ of neighbors mixing a fraction of desired peers with a fraction of randomly selected peers. Let K_p be the number of neighbors peer p will choose. Let $\mathcal{D}_p$ the set of "desired peers" according to p:

$$\mathcal{K}_p = \{q_i\} \cup \{q_j\} \tag{7.1}$$

$$q_i \in \mathcal{D}_p \wedge |\{q_i\}| = \min(|\mathcal{D}_p|, \lceil (1-\alpha)K_p \rceil) \tag{7.2}$$

$$q_j \in \mathcal{N} \setminus (\{q_i\} \cup \{p\}) \wedge |\{q_j\}| = K_p - |\{q_i\}| \tag{7.3}$$

where $\alpha \in [0,1]$ is a coefficient that selects the randomness of the topology: $\alpha = 1$ means a totally random topology and $\alpha = 0$ means a topology just following the "desirableness" of the peers. For the sake of simplicity (7.3) assumes that p knows all the peers in the system, but in actual implementation its "view" will be limited by some peer sampling procedure.

Note that to make the overlay an undirected graph, the neighborhood of peer p must be completed including in it all peers that have selected p as neighbor. This means that the actual neighborhood of P composed by two components: *i)* $\mathcal{K}_p$, i.e., the peers that p has selected and *ii)* the peers that have selected p as neighbor.

$\mathcal{D}_p$ is built following a set of criteria based on network measurements. We envisage a simple, but configurable selection formally expressed as:

$$q \in \mathcal{D}_p : (B_q \geq T_{BW} B_p) \wedge (l_{pq} < T_{RTT}) \wedge (h_{pq} < T_{hop}) \tag{7.4}$$

where B_q is the upload bandwidth of peer q, l_{pq} is end-to-end latency between peers p and q, and h_{pq} is number of IP hops between p and q. T_{BW}, T_{RTT} and T_{hop} are thresholds that define what is a desired peer to connect to. They represent tuning parameters of the system. Basically, the idea is to make each peer connect to a *fraction* $(1-\alpha)$ *of "good" peers, and a fraction* α *of random peers.*

7.4.2 Chunk trading

Chunks are transmitted from peer p to q after p offers to q a set of possessed chunks and q selects one of them.

The negotiation phase is sketched in Fig. 7.1: signaling exchanges are represented by the arrows over the time line in the figure and data transmissions are at the bottom. The negotiation happens through the exchange of offer and select messages. This trading phase have been comprehensively been described in Part I, in particular

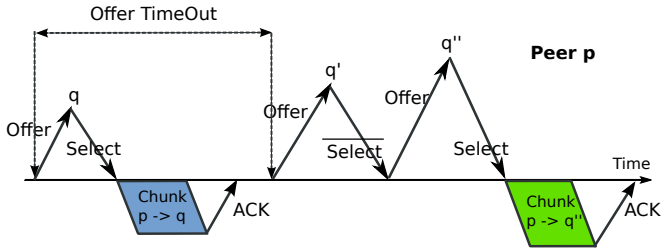

Figure 7.1. Schematic representation of the peer chunk trading mechanism.

in Section . In this case, however, the system is entirely asynchronous so that offers are sent by peer p at a fixed rate λ_p (with exponentially distributed inter-offer times).

Several design choices impact the performance of the trading protocol. In particular: *i)* carefully selecting the value λ_p is essential to efficiently exploit the upload resources while maintaining the transmission queue length small – as described in Chapters 3 and 4; *ii)* the criteria to select peer q to send the offer to; *iii)* the strategy according to which q selects the chunk to download.

In what follows we discuss these choices.

Setting λ_p

Given the average chunk size L, B_p/L is the average number of chunks that can be transmitted between two consecutive offers. To effectively use the peer bandwidth when the system is near overload, it must be:

$$\lambda_p \geq \frac{B_p}{L} \tag{7.5}$$

i.e., the number of offers sent by p must be larger than the chunks it can transmit since not all offers will be followed by a positive select.

Assuming K_p is never a limitation as the topology is well connected, and respecting (7.5), a good selection criterion is

$$\lambda_p = \beta B_p/L \tag{7.6}$$

where $\beta \geq 0$ (typically close to 1) is the *signaling scaling factor*.

Peer selection

The choice of q, the peer to send offers to, can follow different strategies. One simple approach is to select q among neighbors uniformly at random, as proposed in [7].

Chunk selection

For what concerns chunk selection at the receiver, many strategies have been proposed and analyzed in idealized scenarios, like random chunk [5], latest chunk [7], and dynamic deadlines [29]. In this chapter, we use a random chunk selection strategy for its simplicity and robustness and because this is also the choice for many implemented solutions [8, 41]. Moreover, we are focusing on the selection of peers to communicate with based on their attributes, thus keeping the chunk selection strategy simple avoids mingling effects and making results difficult to interpret.

7.5 Performance Evaluation by Simulation

In this section we present a thorough performance evaluation of bandwidth based tuning of the system via simulation, as using bandwidth for real system tuning in PlanetLab experiments is unreliable as explained in Sect. 7.8. All results are obtained with *P2PTV-sim* [18], an open source event driven simulator developed within the NAPA-WINE project.

7.5.1 Network scenario and assumptions

Table 7.2. Characteristics of peer classes.

Class 1:	B1 = 5 Mb/s ± 10%
Class 2:	B2 = 1.6 Mb/s ± 10%
Class 3:	B3 = 0.64 Mb/s ± 10%
Class 4:	B4 = 0 Mb/s, i.e., free riders

Peers are partitioned in four classes according to their upload bandwidth as shown in Table 8.3. We consider four different bandwidth scenarios. Let the parameter H denote the fraction of large bandwidth peers (Class 1 peers). The higher H is, the larger the overall bandwidth heterogeneity will be. The fraction of free-riders (Class 4 peers) is constant and equal to 20%. Fraction of Class 2 and Class 3 peers is then derived by imposing that the system-wide average upload bandwidth is $E[B_p] = 1.3$ Mb/s in all cases. Table 7.3 reports for every scenario the distribution of peers.

The transport network is transparent: it introduces a delay equal to $l_{pq}/2$ to all the datagrams from p to q. The RTTs l_{pq} are measured at the application layer and are proportional to the geodetic distance between peers. Peers are distributed

Table 7.3. Percentage of peers per class for different scenarios.

Class	1	2	3	4
$H = 0.01$	1	76.7	2.3	20
$H = 0.05$	5	58.5	16.5	20
$H = 0.10$	10	35.8	34.2	20
$H = 0.15$	15	13.2	51.8	20

over the Earth surface according to a synthetic model that emulates the distribution of the Internet user population[1]. In particular, peers are scattered over seven domains representing continental/sub-continental geographical regions: Asia (Far East), Europe, Africa, Middle East, Oceania, North America and South America.

Each domain is modeled with disks placed in the center of mass of the relative geographical region (through latitude and longitude coordinates). Their radius are differently calibrated to match the extension of the corresponding geographical region. The mean RTT is $E[l_{pq}] = 182\,\text{ms}$.

We consider scenarios comprising $N = 2000$ peers, if not otherwise indicated. All results are averaged over at least eight independent simulation runs.

7.5.2 Video parameters

In most of the P2P-TV literature so far, a very simplistic synthetic model for the video stream has been adopted according to which the source generates fixed length chunks at a fixed rate. However, compressed video streams are known to exhibit variable and bursty encoding rate, and this can deeply impair the system. Similarly, frames have different importance in the sequence. For example, "intra" frames carry very valuable information (and are therefore bigger), while "inter" frames carry only differential information (and are much smaller). Furthermore, the paramount performance index streaming applications should be the actual Quality of Experience (QoE), which can only be evaluated including the video coding and decoding processes in the simulation. In other words the loss of a frame can have a very variable impact on the QoE, depending on the type of the frame itself. In this chapter we therefore explicitly model the streaming of actual video streams over the P2P-TV system, and evaluate the system performance using a direct measurement of the QoE users get.

The already employed video sequence *Pink of the Aerosmith* is used as benchmark. In this case the length has been cut to 40s, corresponding to 1000 chunks. The spatial resolution is 352×240 pixels, while the time resolution is 25 frame/s.

[1]This distribution has been derived from `http://www.internetworldstats.com/stats.htm`

Table 7.4. Characteristics of the encoded video sequences.

	Pink		
Frame type	Num	Avg. size [kb]	Std. dev
IDR+I	40	79.2	6.43
P	285	57.0	17.10
B	225	38.2	15.3
b	450	27.6	21.7

We selected the H.264/AVC protocol for encoding video sequences. A hierarchical type-B frames prediction scheme has been used, obtaining 4 different kinds of frames that, in order of importance, are: IDR, P, B and b. GOP structure has been set to IDR×8{P,B,b,b}, which can however be violated if the encoder detects a sudden scene change that forces the insertion of a IDR frame. The nominal video rate of the encoder r_s is a free parameter that we vary to enforce different values of the system load defined as $\rho = r_s/E[B_p]$.

Given the highly structured video stream organization, a natural question is how to chop the video data into chunks. We have selected a simple *one to one* mapping according to which every chunk contains exactly one video frame. The source node then generates a new chunk at regular time, i.e., every new frame. This mapping scheme also allows a stricter real-time streaming.

A limit of this scheme is that it is not possible to control the chunk size. As a result, the chunk size distribution is highly variable. As an example, Table 7.4 reports the frame size statistics for the video benchmarks when $r_s = 1$Mb/s.

To assess video quality at the receiver we start from the classical Peak Signal-to-Noise Ratio (PSNR) metric [22, 42]. PSNR is a widely adopted objective video quality index that provides the mean square error between the original video and the received one. Note that the PSNR scale is logarithmic in dB, so that a difference of 2dB corresponds to a very large improvement of the QoE. For example, from Fig. 7.6 doubling the encoder rate from $r_s = 780$kb/s to $r_s = 1410$kb/s improves the PSNR by 2.5dB only (from 42.91dB to 45.45dB).

For each peer, the computation of the PSNR is done on a frame by frame basis, comparing the original image to the one that is reconstructed at the receiver. The obtained values are then averaged over all frames, obtaining the "PSNR per peer". Finally, the "overall average PSNR" is obtained by averaging the PSNR over all peers. The computation of the PSNR cannot be done in correspondence of a missing frame. In this latter case, we assume that the receiver uses the last correctly decoded frame as reference to compute the PSNR. The PSNR at the receivers includes both the effect of the encoder and of chunk loss occurred during the P2P-TV distribution, so it must always be compared with the Encoded Video Quality (EVQ), i.e., the

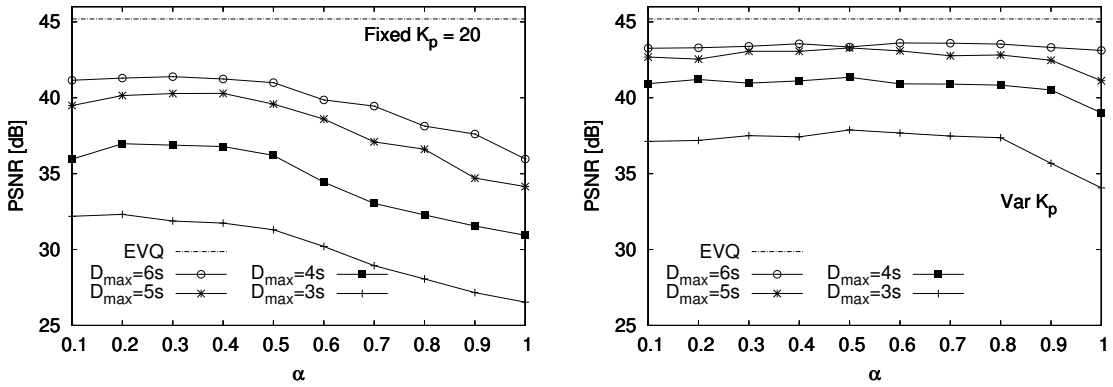

Figure 7.2. Average PSNR versus α for different values of the playout delay with $\rho = 0.9$ and $H = 0.10$ for fixed and variable K_p.

PSNR of the encoded sequence.

7.5.3 Results

First, we wish to assess the effectiveness of the proposed latency/bandwidth-aware overlay topology design. We set $T_{RTT} = E_q[l_{pq}]/2$ (half the mean RTT), and $T_{BW} = 1/2$ (peers are desired by p only if they have at least half the upload bandwidth of p itself). $T_{hop} = 0$ is not used since the simulator does not represent such details.

Impact of the neighborhood size K_p

The value of K_p (number of connected peers) can be conveniently set based on the upload bandwidth of p, as only high bandwidth peers need large neighborhoods to effectively exploit their bandwidth. Given the absence of peer sampling protocol in the simulator, and the fact that every peer knows all other peers' characteristics, the topology is built letting each peer select

$$K_p = \max\left(3, \lceil \delta B_p / r_s \rceil\right) \tag{7.7}$$

peers at random following Eqs. (7.3) and (7.4) and opens bidirectional links with them. Another possibility is setting simply K_p constant.

Fig. 7.2 compares the two possible policies of static K_p and bandwidth aware K_p, plotting the average PSNR versus the overlay construction parameter α, for different choices of the playout delay D_{max} buffer. The simulations refer to a network scenario with $H = 0.10$. The *Pink* video sequence encoded at rate $r_s = 1168$kb/s, so that the average offered load is $\rho = 0.9$.

The left plot refers to $K_p = 20$ fixed for every peer in the network. In this case the actual neighborhood size is on average equal to 40 as the inbound selection of

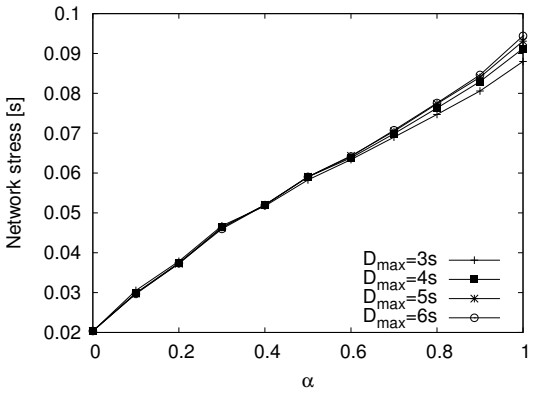

Figure 7.3. Network stress versus α, for different topologies and values of the playout delay, $\rho = 0.9$.

p by other peers leads to a geometric distribution with average K_p. The right plot, instead, refers to the case in which K_p is set based on the bandwidth as in (7.7) with $\delta = 10$ and $\beta = 2$. In this case, the average neighborhood size is measured a posteriori equal to 28.3, but it varies greatly with the bandwidth class. The Encoded Video Quality (EVQ) at the source is reported as reference. The effects introduced by the distribution system on the perceived quality can be grasped by comparing the received PSNR with the PSNR at the source. The advantage of dynamic K_p is evident. This confirms that exploiting upload capacity of high-bandwidth peers plays a key role as already shown in [15].

Impact of α

The parameter α (that represents the randomness in choosing neighbors) has an impact on the video perceived by users. Always referring to Fig. 7.2, the performance of the system is quite good for all $\alpha \in [0.1,0.5]$. Values of α closer to 1 worsen the performance for effects of the larger latencies among neighbor nodes, especially in the case of fixed K_p and/or when a tight play-out buffer is enforced.

Turning our attention on the underlying transport network cost, Fig. 7.3 reports the *network stress*, i.e., the average distance covered by chunks expressed in terms of delay between transmitter and receiver. This metric can be taken as the cost paid by the transport network for carrying traffic related to the P2P-TV service.

The network stress is monotonic increasing and almost linear with respect to α. By decreasing α, i.e., better localized neighborhoods, the stress for the underlying network is reduced.

Fig. 7.4 explores the effect of α in different bandwidth scenarios. The curves refer to the case of variable K_p with play-out delay $D_{\max} = 5$ s. In all cases performance

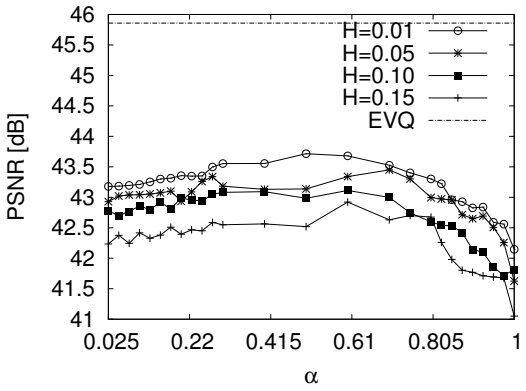

Figure 7.4. Average PSNR versus α for different values of H with $D_{max} = 5$s.

consistently improves by reducing α from 1 to about 0.7, then it remains substantially flat until $\alpha = 0.2$ and then slightly decrease until the topology degenerates for $\alpha = 0$ (not shown because the topology is disconnected).

Given these results, a choice of small values of α is highly supported. In the following, we choose $\alpha = 0.1$ as a good trade-off between overlay robustness and performance benefits.

Fig. 7.5 reports the overall PSNR versus D_{max}, considering a scenario with $N = 2000$ and $N = 7000$ peers. Results show the beneficial impact of reducing α, which is consistent for several playout delays. As expected, to achieve similar PSNR performance, the playout delay needs to be increased when the number of peers in the system increases. This is due to the larger overlay topology diameter (recall that the number of hops chunks have to be transmitted grows logarithmically with N), which translates into higher delay. Therefore, the benefit of a smart overlay design is higher for large N too.

7.5.4 Impact of r_s and β

We consider only the location/bandwidth-aware policies with variable degree and $\alpha = 0.1$, since these choices achieve globally the best performance.

Fig. 7.6 reports the received average PSNR versus the network load ρ (on the bottom x-axis) or equivalently the video source rate r_s (on the top x-axis). Different values of β are shown to calibrate λ_p. First, notice that the PSNR increases initially for increasing video rate, reflecting the higher quality of the encoded stream. However, for $r_s \geq 1.03$Mb/s (i.e., $\rho \geq 0.8$), dramatic performance degradations are suffered: the increased system load causes larger chunk loss rates.

Considering the impact of β on the system performance, we notice that too small values of β limit the high bandwidth peers ability to exploit their upload

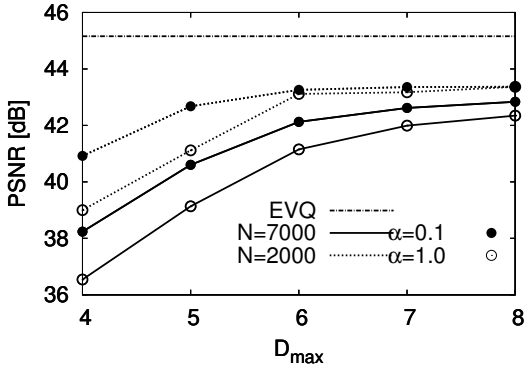

Figure 7.5. Average PSNR for different values of the playout delay and number of peers with $\rho = 0.9$ and $H = 0.10$.

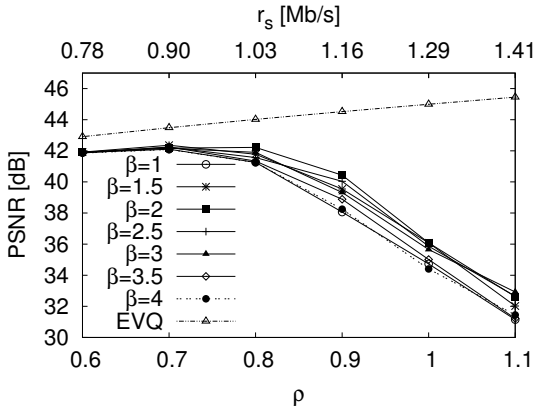

Figure 7.6. Average PSNR versus ρ for different values of β with $D_{max} = 5$s, $\alpha = 0.1$. The corresponding video rate r_s is reported on the top x-axis.

capacity, causing undesirable chunk losses independent of the playout delay which is set $D_{max} = 5$s. Too large values of β make on the contrary chunk delivery delay increase due to queuing at the transmitter peer, causing losses due to inadequate playout delay. In our setting, the best choice of β is 2.

7.6 System Implementation

All algorithms presented through this part of the document have been implemented in a real P2P-TV client that have been successfully developed in the NAPA-WINE project, the so called *PeerStreamer* [14].

PeerStreamer is a general client for unstructured P2P-TV systems. It has been

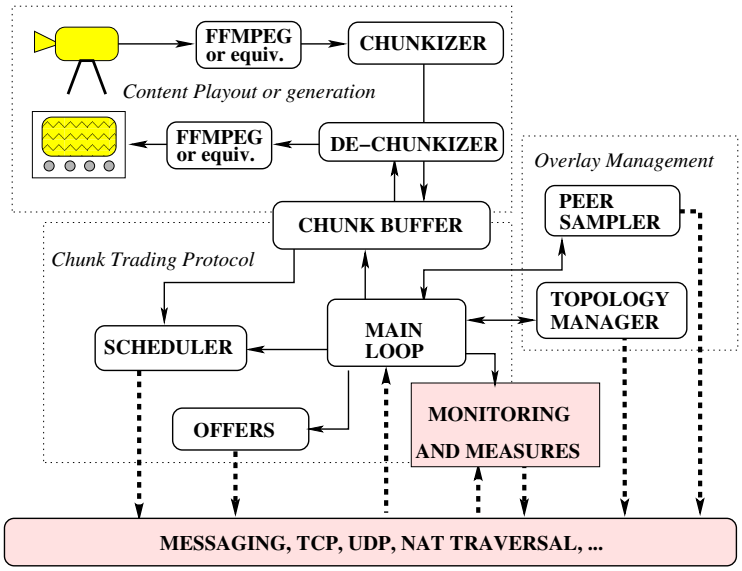

Figure 7.7. PeerStreamer peer architecture.

designed and developed starting from GRAPES [43], a set of C libraries implementing basic building blocks for P2P streaming applications which allow to easily develop and prototype different algorithms. In this chapter, we explicitly focus on the implementation of the system described in Sect. 7.3 and 7.4 using a fully distributed solution.

Fig. 7.7 sketches the logic and modular organization of PeerStreamer. A gossiping protocol maintains the overlay network among peers; the overlay is the support for negotiation and chunk transmission. All communications are over UDP, due to the real time nature of the application.

The source peer implements the *Chunkiser* to process the media stream (e.g., an encoded file, or a live MPEG TS stream coming from a DVB-T card, or the raw video of a webcam) and produces the sequence of chunks according to the simple one-frame → one-chunk mapping (this choice can be changed with a simple parameter obtaining larger chunks and less signaling overhead). The Chunkiser is implemented using the ffmpeg[2] libraries, so that several different codecs (for example, MPEG video, theora, H.264, ...) are supported.

The chunks obtained from the Chunkiser are pushed in the *Chunk Buffer*, becoming available to the *Chunk Scheduler* and to the *Chunk Trading Protocol* to be redistributed among the neighbors.

PeerStreamer main loop implements the global application logic it is responsible for: *i)* handling the chunk buffer, *ii)* triggering state information exchange with

[2] `http://ffmpeg.org`

the neighbors, *iii)* sending and receiving chunks, *iv)* running the main protocols needed for the streaming. Some of these activities are performed in a cyclic way, while some other activities are performed reacting to external events (such as the arrival of a network packet). As a result, the main loop implements two distinct (but interacting) execution flows, corresponding to an active behavior (the activities which are cyclically executed) and a passive behavior (the activities executed in response to external events). The combination of the active thread and the passive thread allows to implement the offer-select protocol, which defines the actions taken at the peer during the whole streaming session. In the implementation, the offer-select protocol and the chunk trading mechanisms strictly follow the one described in Sect. 7.4.2.

Finally, the PeerStreamer architecture allows each peer to obtain a set of measurements thanks to the "Monitoring and Measures" module. It has two modes of operation: passive measurements are performed by observing the messages that are exchanged anyway between two peers, e. g., when exchanging video chunks or signaling information; active measurements, in contrast, craft special probe messages which are sent to other peers at the discretion of the monitoring module. The design of this monitoring module is one of the most innovative goals of NAPA-WINE since it exposes vital information to design network friendly P2P applications. Measurements are available at the chunk and packet levels. Several network layer metrics are monitored: *i)* delay between peers (e.g., RTTs, Delay Jitter), *ii)* loss probability, *iii)* path capacity and available bandwidth, *iv)* number of hops traversed. Thanks to a simple and modular architecture, measurements can be added as (compile-time) plug-ins, and activated on demand. The monitoring layer is implemented at every peer, and information it collects is made available to the all peers via signaling primitives. In this chapter, we rely on both RTT and upload capacity measurements. The upload capacity estimation is performed using packet-pair based technique, i.e., by passively observing the inter-packet gap distribution of packets received by neighboring nodes. RTT estimation is simply achieved by timestamping signaling messages.

7.7 Overlay Management

To demonstrate the feasibility of the overlay topology design algorithms described in Sect. 7.4, we explicitly adopted a fully distributed solution. While this choice makes the problem more challenging, it permits to prove that the proposed policies can be implemented in a fully decentralized manner as typically requested in purely P2P systems.

In more detail, peers establish contacts with several other peers by means of a gossipping protocol based on Newscast [44]. Each peer periodically advertises to

one of its neighbors the list of peers he knows and their estimated upload capacity. The reception of a Newscast message allows to discover other peers *i)* identity, and *ii)* upload capacity. Thanks to the monitoring module, the peer passively estimates the RTT with any neighbor when some signaling information is exchanged, so that $\mathcal{D}_p$ is build. The selection of peers to include/exclude in the neighborhood $\mathcal{K}_p$ is done targeting a fraction α of desired neighbor peers.

7.8 Performance Evaluation by Experiment

Running experiments of video streaming over the Internet with a large number of peers requires setting up a proper environment which is not an easy task. We decided for a mixed environment where some peers are in our University Campuses and others are run on the public PlanetLab nodes scattered around the world. We selected about 400 PlanetLab nodes choosing the one with the highest uptime. The video source is at the University of Trento Campus, and it is responsible to seed the swarm by injecting 3 copies of each video chunk, sending them to randomly chosen peers within the neighborhood. For the experiment, we used the same Pink video encoded at 600 kb/s to avoid the bandwidth limitation imposed by the PlanetLab policies. The source is looping through the video and it is always up and running.

For each experiment, each peer enters the swarm at a random time in the first 30s. Then, it participates to the swarm for a period of time uniformly distributed between [300 : 360]s, then it leaves the swarm and reconnects again after a uniform random idle time in [50 : 80]s. The churning level is therefore quite high. Measurements reported in the following have been collected considering a period of time of 5 min after an initial transient of 10 min.

7.8.1 Bandwidth and delay measurements in the experiments

Fig. 7.8 reports the RTT (on left plot) and upload capacity (on right plot) measured among peers during one experiment. About 30% of all peer-to-peer pairs are present in this dataset, since peers do not exchange packets with all other peer. Considering RTT, the tail of the distribution beyond 1 s is very long, showing that communication with PlanetLab nodes is often impaired. Measurements reflect the typical worldwide distribution of PlanetLab nodes, i.e., nodes in the same or different network, region or continent exhibits different RTT according to the propagation delays. Upload capacity measurements confirm that nodes involved in the experiment can exploit high capacity connectivity. However, 65% of paths shows that the end-to-end upload capacity is smaller than 10Mb/s, with 19% of paths suffering severe congestion so that upload rate is negligible (and possibly RTT is very large). Note that the network

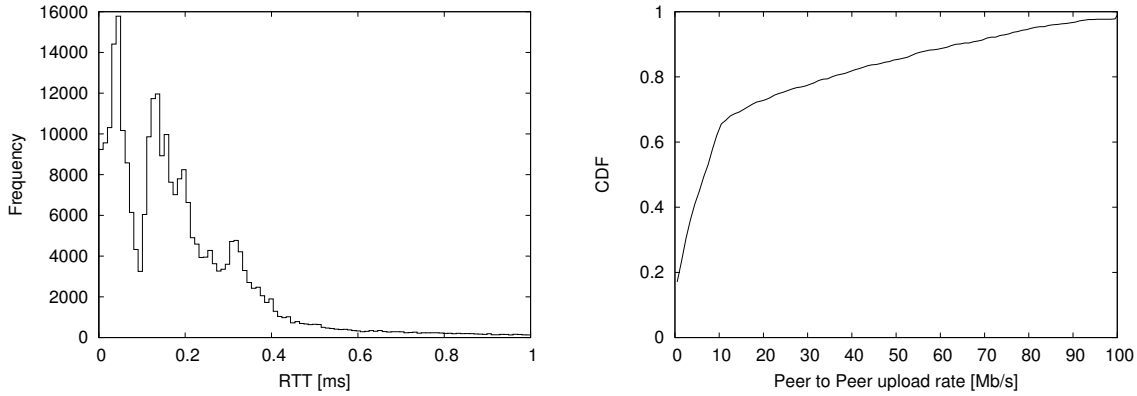

Figure 7.8. Histogram of the pair-wise RTT (left plot) and CDF of end-to-end upload capacity (on the right) measured during experiments.

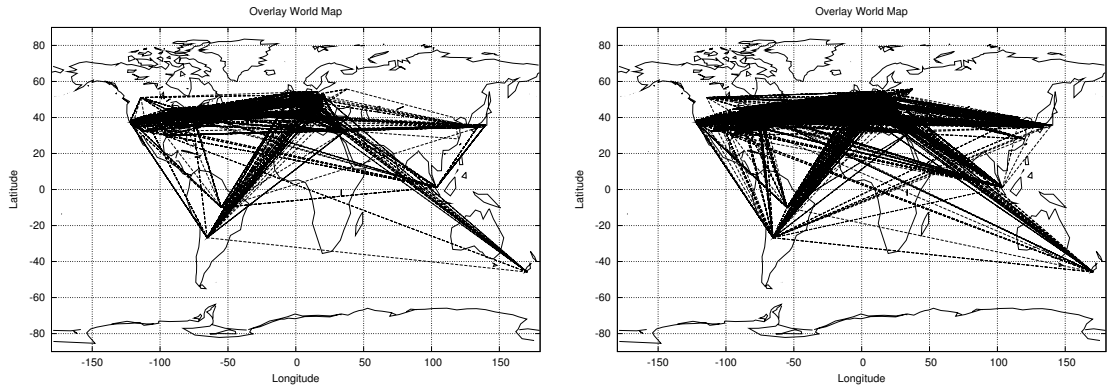

Figure 7.9. Example of overlay topologies for $\alpha = 10$ and $\alpha = 100$ for $T_{RTT} = 50$ms.

scenario augmented by the typical random load observed in PlanetLab nodes (which are used for many experiments at a time) poses critical condition to the experiment.

7.8.2 Experimental results

With reference to (7.4), we selected $T_{BW} = 0.1$, $T_{hop} = 30$ and $T_{RTT} \in \{50,100,150\}$. This choice mitigates the peculiarity of the scenario we consider in which the upload capacity of nodes is normally not related to physical limitation but either to congestion or to explicit shaping, while the hop count distance among nodes is either very large or very very small due to nodes in the same institution, thus we practically exclude the hop count based selection (this is also coherent with the simulation scenario).

Fig. 7.9 compares two topology snapshots that have been obtained considering $\alpha = 0.1$ and $\alpha = 1$ on the left and right plot, respectively. In particular, it shows

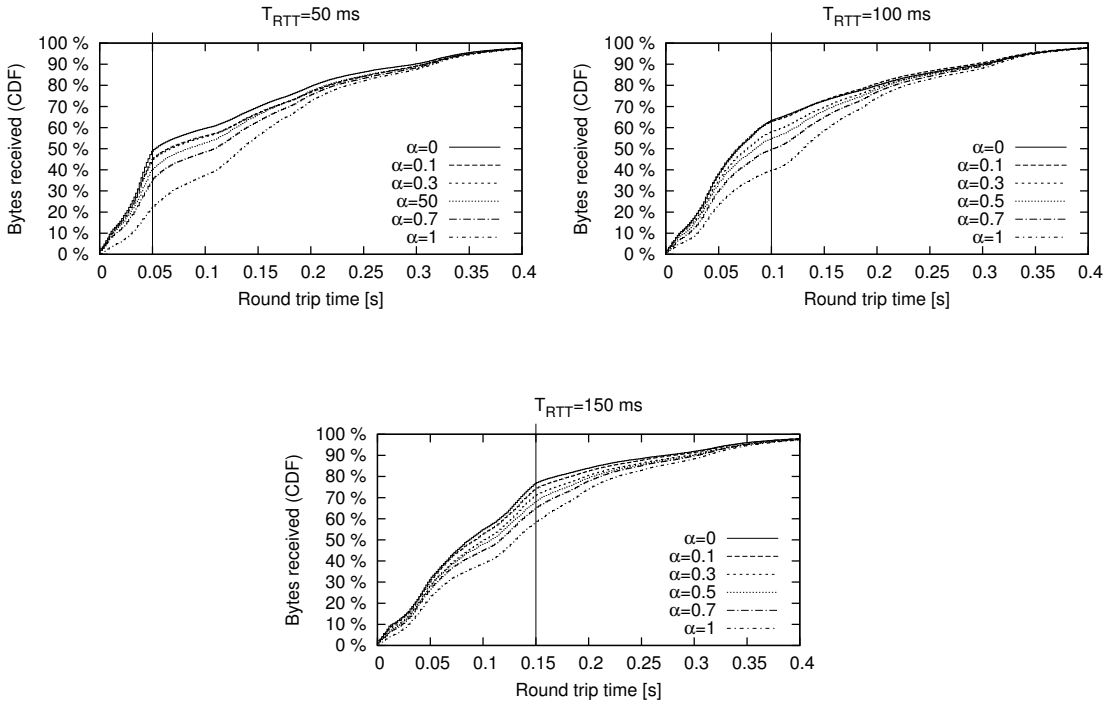

Figure 7.10. Number of bytes received by peers as a function of observed RTT on the overlay connection

the 5% of the overlay link that carried the largest amount of traffic. Albeit not extremely evident, the topology localization can be observed even on this qualitative representation.

To quantify the intuition above, Fig. 7.10 reports the fraction of bytes downloaded by all the peers in the swarm as a function of the RTT measured between the peer sending the chunk and the peer receiving the chunk. The upper left plot is for $T_{RTT} = 50$ ms, the upper right one for $T_{RTT} = 100$ ms and the lower one for $T_{RTT} = 150$ ms. Curves for α =0, 0.1, 0.3, 0.5, 0.7, 1 are plotted using different lines. Consider first the case $T_{RTT} = 50$. Larger fractions of bytes received from nearby (small RTT) peers means more localized traffic and hence a smaller cost for the network.

Results show that for smaller values of α the amount of traffic received from peers with small RTTs is larger: by increasing the fraction of "desired" peers in the neighborhood the traffic is more and more local. Considering results for different choice of T_{RTT}, the qualitative behavior remains the same, but the distribution weight moves toward larger RTTs, with the distribution 'knee' correlated with the value of T_{RTT}.

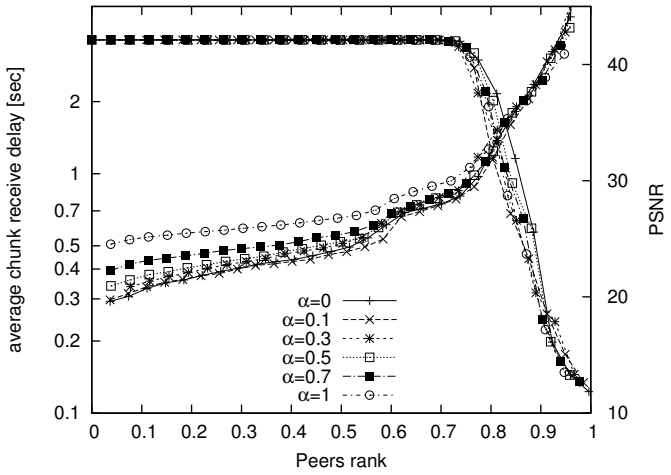

Figure 7.11. average receive delay of chunks (left axis, increasing curves) and PSNR of received video (right axis, decreasing curves) for $T_{RTT} = 100\,\text{ms}$ and different values of α.

Fig. 7.11 presents instead the perspective of the users, presenting the average PSNR and the chunk delivery delay together in the same plot. The PSNR curves (right y axis) show that the quality of the received video is very good for nearly 80% of peers, regardless of the localization parameter α. Given the uncontrolled experimental setup, it is impossible to analyze the reason why the remaining 20% of peers have bad quality. However, looking at the delivery delay of chunks (the average is reported here) it is clear that these peers experience also very high delivery delays, so that this can be ascribed, with high probability to the high RTTs measured and reported in Fig. 7.8, due either to host overload or to Internet congestion. The curves relative to the delivery delay also show that more localized neighborhoods improve performance and not only reduce network costs, with the exception of $\alpha = 0$ where the overlay topology has bad connectivity characteristics. Moreover, we can observe exceptionally good delay performance, with nearly 80% of the peers (in practice all the peers that receive all the video) receiving chunks with an average delay of less than 1 s, which can easily empower playout delays at the application level smaller than 4–5 s.

7.8.3 Further experiments in an emulated context

In addition to the results on PlanetLab we decided to run some experiments in an controlled scenario having full control of the network parameters. The experimental set-up is composed of around 1000 PCs running our PeerStreamer in labs at Politecnico di Torino. The network is using TC, the standard Linux Traffic Controller daemon. We added both upload bandwidth constraints to every peer as well as

delays. The scenario we consider is similar to the one considered in Sect.7.5, in which peers are partitioned in four classes according to their upload bandwidth (see Table 8.3)[3]. The average bandwidth in the system is about 1.5 Mbit/s. Similarly to the simulated scenario peers are grouped in four logical domains comprising a different number of PCs; latencies between peers are set to few tens of ms when both peer lie in the same logical domain, and to more than 100 ms when the peers lie in different logical domains. The overall average latency is $E[l_{pq}] = 70$ms. In the experiments we have used an absolute bandwidth threshold $T_{bw} = 1.0$ Mbit/s and a RTT threshold $T_{RTT} = 60$ ms.

Fig. 7.12 (left plot) shows the fraction of video-bits received by every individual peer for different values of α in the case in which average video with rate is 1 Mbit/s. Observe that the network load is rather high in this scenario (around $\rho = 0.85$ considering the signaling too).

Observe how the performance dramatically depends on α. For intermediate values of $\alpha = 40$,60,80 we get the best performance. Values of α which are too small leads to poor performance, because in these cases the topological properties are poor. The totally random topology also performs rather poorly, because in this case the system is penalized by the long latencies and by the fact that large upload bandwidth of class 1 peers not efficiently exploited. In this critical bandwidth scenarios the beneficial impact of a smart topology management based on measurements is even more significant in the emulated scenario with respect to what we have observed in simulation. At last Fig. 7.12 (right plot) shows the effect of α on traffic localization. The plot refers to the case in which the average rate of the distributed video is reduced to 0.8 Mbit/s, and losses are negligible for all the selected values of α. This permits to fairly compare plot for different values of α. Observe as already intermediate values of α such as $\alpha = 60$ permit a significant reduction of the network cost with respect to network unaware solution.

[3]we notice, however, that the upload bandwidth of free loaders has been set to 200 kbit/s to allow them to exchange signalling messages.

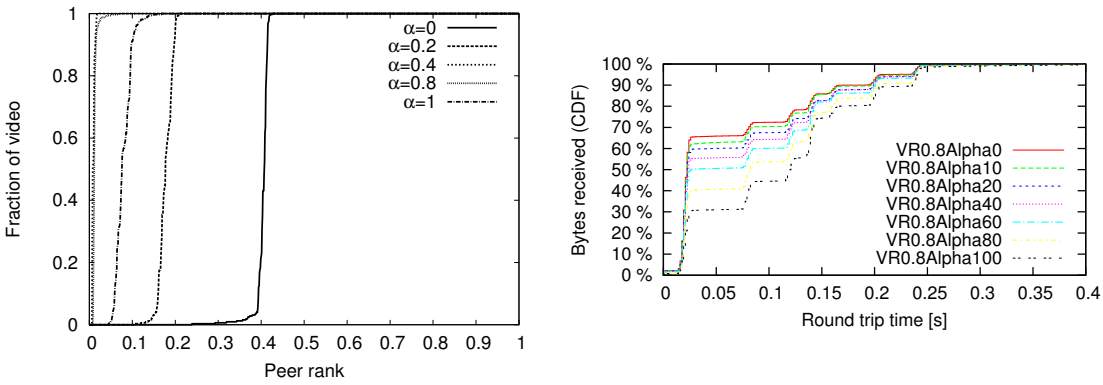

Figure 7.12. Fraction of received bits for different values of α (left plot), and CDF of received bytes as a function of observed RTT for different values of α in the emulated scenario (right plot).

Chapter 8

Experimental comparison of neighborhood filtering strategies

8.1 Introduction

As already said in Chapter 6 a large body of research work has focused on the design and analysis of efficient algorithms for the overlay topology construction and maintenance. Most of the previous works, however, have mainly a theoretical flavor, thus performance analysis of different proposed strategies have been carried out in rather idealized scenarios exploiting simulations or mathematical models. Few works undergo implementation and present actual experiments, and even those are usually limited to few tens of peers. A detailed discussion of related work is presented in Sec. 8.2.

What is still missing is a systematic comparison of different algorithms in an actual and known environment. Indeed, only an actual implementation allows to fully evaluate the different policies, assessing the impact of signaling, measurements, implementation issues, etc. This chapter tries to fill this gap, providing a comprehensive and purely experimental comparison of different strategies for the construction and the maintenance of the overlay topology for P2P-TV systems.

The algorithms we investigate are all based on the process of selection of neighbors which is run by every peer, keeping the system fully distributed and without the need for external help, or a centralized 'oracle' to help peers. Algorithms are based on a *selection* and *replacement* criteria, according to which each peer chooses the peers he would like to have as parents (i.e., peers from which download chunks). Blacklisting feature allows to avoid cycling among bad parents. Overall, we explore 12 different combination of criteria (24 if blacklisting is enabled), based on metrics such as Round Trip Time (RTT), peer upload capacity, number of received chunks,

etc. Intuitively, these are metrics that are known to either i) favor traffic localization, e.g., choosing peers with smaller RTT, or ii) improve system performance, e.g., choosing peers with larger upload capacity.

We test these algorithms in three network scenarios in which we control peer upload capacity, end-to-end RTT and packet loss. In the simplest scenario, peer upload capacities are heterogeneous among peers, while RTT forms 4 clusters, with inter-cluster RTT being smaller than intra-cluster RTT. Then we consider a biased upload capacity distribution, where high capacity peers are all in the same cluster. Next, we add the impact of eventual packet loss on long-distance paths among clusters, facing an almost adversarial scenario.

Results show that policies that select peers based on network distance coupled with policies that drop peers based on their contribution are performing well in all scenarios.

Finally, we wish to emphasize that, for the first time to the best of our knowledge, we present reproducible experimental results for a fully controlled and publicly available real implementation of a P2P-TV system referring to a rather large scale set-up comprising 1000 peers. All the software we have developed is Open Source and released under the (L)GPL.

8.2 Related Work

Many popular commercial applications such as *PPLive, SopCast, Octoshape* were proposed in recent years, but their creators did not publish any information about their internal implementation, making impossible any statement about their overlay topology design strategies. Focusing on available literature on purely mesh-based P2P-TV streaming systems, many solutions can be found, but also in this case, to the best of our knowledge, none of them provides *general and detailed* guidelines for the overlay topology design process. An early solution called *GnuStream* was presented in [45]. Based on *Gnutella* overlay, *GnuStream* implemented a simple load distribution mechanism: peers were expected to contribute to chunks dissemination in a way proportional to their current capabilities. A more refined solution called *PROMISE* was introduced by Hefeeda et al. in [46]. Even if PROMISE proposed an improved seeder choice based on network tomography techniques, peers were interconnected through *Pastry* overlay topology which implements - as many others P2P substrates like Chord [47] or *CAN* - some naive location awareness based on number of IP hops. *DONet* (or *Coolstreaming*) [11] is a successful P2P streaming system implementation. This system employs a scheduling policy based on chunk rarity and available bandwidth of peers, but its data-driven overlay topology does not exploit any information coming from underlying network levels. In [48] the brand new implementation of Coolstreaming application is presented. Many new features were

introduced to improve the streaming service and, in particular, authors proposed a new neighbor re-selection heuristic based only on peers uplink bandwidth. In [49], authors showed the design aspects related to their application called *AnySee*. Even if partially based on multicast, this hybrid mesh-based system relies on an overlay topology that aims at matching the underlying physical network while pruning slow logical connections. However, no deep investigation about performance of their overlay design strategy is provided. In [50] authors presented a study about some key design issues related to mesh-based P2P-TV systems. In particular they focused their attention on understanding what are the real limitations of this kind of applications and presented a system based on a directed and randomly generated overlay. Some fundamental improvements were introduced: e.g., the degree of peers' connectivity proportional to their available bandwidth.

Turning our attention on more theoretical studies about the overlay topology formation and maintenance, in [33] the problem of building an efficient overlay topology, taking into account both latency and bandwidth, has been formulated as an optimization problem; however, the complex interactions between overlay topology structure and the chunk distribution process are completely ignored in [33], where continuous streams of information are distributed (in a purely push fashion) among peers.

In [16] a theoretical investigation of optimal topologies is formulated, considering latency and peer bandwidth heterogeneity; scaling laws are thus discussed. In [17], a distributed and adaptive algorithm for the optimization of the overlay topology in heterogeneous environments has been proposed, but network latencies are still ignored. The authors of [34] propose a mechanism to build a tree structure on which information is pushed. They combine two ideas: good topological properties are guaranteed by means of prefixes based on peers identifiers (similarly to what is done in other structured P2P systems) and latency awareness is used to select a specific peer between those with the same prefix. Similar in spirit, but in unstructured systems, we propose in this chapter an overlay topology design strategy that, taking into account latency and peer heterogeneity, aims at creating an overlay with good properties and low chunk delivery delays. In highly idealized scenarios, [35] shows with simple stochastic/fluid models that overlay topologies with small-world properties can be effectively exploited to support chunk distribution in P2P-TV systems.

Finally, it is worth to mention [37], where the authors experimentally compare the performance of unstructured systems and structured, multiple-tree based systems. Results in [37], indicates that unstructured systems tends to outperform tree-based systems in highly dynamic scenarios as well as in scenarios characterized by bandwidth limitations, which strengthen our choice of exploring topology management policies for mesh-based streaming systems.

8.3 PeerStreamer Description

Empowering this work is the already cited PeerStreamer [14]. PeerStreamer is an Open Source P2P-TV client that stems from the developments and research of the NAPA-WINE EU project [13] whose overall architecture and vision are briefly described in Section 7.6 and detailed in
citenapa-arch10.

The logic and modular organization of PeerStreamerhas already been described earlier in this document in Section 7.6 and in Fig. 7.7. However, some specific modules have been modified with respect to the implementation shown in Section 7.6. For briefness, in following sections (Sections 8.3.1 and 8.3.2) we only report the main modifications we applied:

- we introduced in this implementation Hose Rate Control (HRC) scheme that has been presented in Chapter 4;
- the overlay management module (detailed in Sect. 8.3.2), has been deeply modified to enable a more effective and flexible neighbors selection mechanism.

8.3.1 Hose rate control

PeerStreamer is based on a chunk-based stream diffusion. Peers that own some chunks offer a selection of them to some destination peers in their neighborhood. The receiving peer acknowledges the chunks he is interested into, thus avoiding multiple transmissions of the same chunk to the same peer. The negotiation and chunk transmission phase is sketched in Fig. 8.1: signaling exchanges (Offer/Select messages) are represented above the time line and chunk transmissions (colored blocks) are below the time line. The *number of offers* per second a peer sends plays a key role in performance. Intuitively, it should be large enough to fully exploit the peer upload capacity, but it must not be so large to cause the accumulation of chunks to be transmitted adding queuing delay before to chunk transmissions. In Chapter 4 we proposed Hose Rate Control (HRC) to automatically adapt the number of offers to both peer upload capacity and system demand. For this set of experiments we adopt HRC: simpler trading schemes are less performing and can hide the impact of the overlay on the overall performance of the system.

For chunk scheduling, offers are sent to neighbors in round-robin. They contain the buffermap of the recent chunks the peer possesses at that time. After receiving an offer, a peer selects one chunk based on a "latest useful" policy sending back a select message. The latest useful policy means the peer selects the most recent chunk it does not have. This has been proven optimal for streaming systems with centralized scheduling in [51].

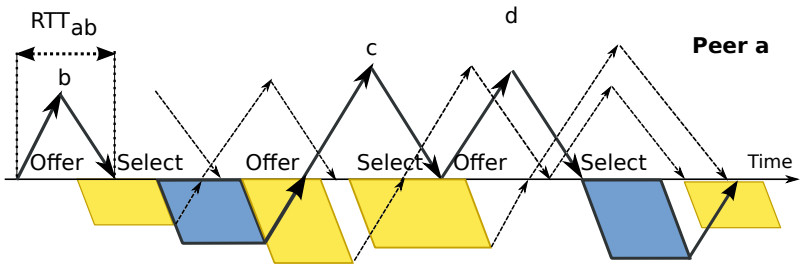

Figure 8.1. Pictorial representation of chunk exchanges from one peer with the offer-select protocol used by PeerStreamer.

The source acts as a standard peer, but it does not participate in the offer/select protocol. It simply injects one or more copies (5 in our experiments) of the newly generated chunk into the overlay.

8.3.2 Overlay management

The approach for building the overlay topology in PeerStreamer is fully distributed: each peer builds its own neighborhood following only local measures, rules and peer sampling. The overlay topology is represented by a directed graph in which the peer at the edge head receives chunks from the peer at the edge tail, which is the one sending offers. Each peer p handles thus an "in-neighborhood" $\mathcal{N}_I(p)$ and an "out-neighborhood" $\mathcal{N}_O(p)$. $\mathcal{N}_I(p)$ collects all peers that can send chunk to p (p parents); $\mathcal{N}_O(p)$ collects all peers that can receive chunks from p (p children). Alternatively, $\mathcal{N}_I(p)$ is the set of peers that *offer* p new chunks; while p *offers* its chunks to peers in $\mathcal{N}_O(p)$. Distinguishing between $\mathcal{N}_I(p)$ and $\mathcal{N}_O(p)$ guarantees a greater flexibility in topology management than algorithms that impose the reciprocity between peers. The overlay topology $\mathcal{T}_S$ is the union of all the in-neighborhoods

$$\mathcal{T}_S = \bigcup_{p \in \mathcal{S}} \mathcal{N}_I(p) \tag{8.1}$$

where $\mathcal{S}$ is the set of all the peers in the swarm[1].

Referring again to Fig. 7.7, the topology management is split into two separate functions. The *peer sampler* has the goal of providing p with a stochastically good sample of all the peers in $\mathcal{S}$ along with their properties; PeerStreamer implements a variation of Newscast [44] for this function. The *neighborhood manager* realizes the task of filtering the peers that are deemed the most appropriate for interaction. Filtering is based on appropriate metrics and measures and it is the main focus of this chapter.

[1]Notice that since $\mathcal{N}_O(p)$ are built passively, they do not contribute to construction of the swarm topology.

8.4 Neighborhood and Topology Construction

In PeerStreamer every peer p selects other peers as in-neighbors and establishes a management contact with them. Thus each peer p actively selects parents to possibly download chunks when building the set $\mathcal{N}_I(p)$. Similarly, p passively accepts contacts from other peers that will form the set $\mathcal{N}_O(p)$ of children. There is no limitation to $N_O(p)$[2].

Every peer p manages a blacklist of peers in which it can put peers that were perceived as very poorly performing parents. Peers in the blacklist cannot be selected for inclusion in $\mathcal{N}_I(p)$. Blacklisted peers are cleared after the expiration of a time-out (set to 50 s in the experiments).

The size N_I of $\mathcal{N}_I(p)$ is equal for every peer p: its goal is to guarantee that p has enough parents to sustain the stream download with high probability in face of churn, randomness, network fluctuations, etc. The size $N_O(p)$ of $\mathcal{N}_O(p)$ is instead a consequence of the filtering functions of the peers that select p as parent. The goal is to let the dynamic filtering functions of peers $q \in \{\mathcal{S} \setminus p\}$ select $\mathcal{N}_O(p)$ in such a way that the swarm performances are maximized. For example, peers with higher upload capacity should have larger number of children than peers with little or no upload capacity.

The update of neighborhoods is periodic, maintaining the topology dynamic and variable, so that churn impairment is limited, and the swarm can adapt to evolving networking conditions. In particular, every T_{up} seconds each peer p independently updates $\mathcal{N}_I(p)$ *dropping* part of the old in-neighbors while *adding* fresh new parents. Two parameters are associated to this scheme: the update period T_{up} and the fraction F_{up} of peers in $\mathcal{N}_I(p)$ that is replaced at every update. The add operation guarantees $\mathcal{N}_I(p)$ has size N_I (if at least N_I peers are known). If not otherwise stated $N_I = 30$, $T_{\text{up}} = 10\,\text{s}$ and $F_{\text{up}} = 0.3$. The latter two values result in a good compromise between adaptiveness and overhead. Their choice is robust and we omit sensitivity analysis due to lack of space.

8.4.1 Metrics driving the neighborhood selection

At every update, $\mathcal{N}_I(p)$ is the result of two separate filtering function: one that selects the peers *to drop*, and another one selecting parents *to add*. For these filtering function we essentially consider both simple network attributes such as peer upload bandwidth, path RTT or path packet loss rate, and some application layer metrics, such as the peer offer rate[3] or number of received chunks from a parent.

[2] In the actual implementation $N_O(p)$ is limited to 200 peers, but the limit is never reached.

[3] HRC adapt the peer offer rate to peer upload capacity. It can thus be seen as an indirect measure of its available upload bandwidth.

Some metrics are static *peer metrics*: once estimated, they can be broadcast with gossiping messages and are known *a-priori*. Other metrics instead are *path attributes* between two peers and must be measured and can only be used as *a-posteriori* indicators of the quality of the considered parent as perceived by p.

Both add and drop filtering functions are probabilistic to avoid deadlocks and guarantee a sufficient degree of randomness. Considering any metric, we assign a selection probability w_q to every candidate q as

$$w_q = m_q / \sum_{s \in \mathcal{N}_S(p)} (m_s)$$

where m_q is the metric of q and $\mathcal{N}_S$ is either $\mathcal{N}_I$ for drop filtering or the set of candidate parents for add filtering.

8.4.2 Add filters

We consider the following four criteria to add new parents:
RND: Candidate neighbors are selected uniformly at random. $\forall q, m_q = 1$;
BW: Candidate neighbors are weighted according to their upload bandwidth C_q. $\forall q, m_q = C_q$;
RTT: Candidate neighbors are weighted according to the inverse of the RTT between p and q. $\forall q, m_q = 1/RTT_q(p)$; If $RTT_q(p)$ is still unknown, $w_q = 1s$[4].
OFF: Candidate neighbors are weighted according to the rate they send offer messages R_q. $\forall q, m_q = R_q$;

8.4.3 Drop filters

For what concerns the criteria to select neighbors to be dropped, we consider:
RND: Neighboring peers are dropped uniformly at random. $\forall q, m_q = 1$;
RTT: Neighboring peers are dropped with a probability directly proportional to the RTT between p and q. $\forall q, m_q = RTT_q(p)$;
RXC: Neighboring peers are dropped with a probability proportional to the inverse of the rate at which it transferred chunks to p. $\forall q, m_q = 1/RXC_q(p)$. This metric essentially assigns a quality index related to the parent ability to successfully transfer chunks to p.

8.4.4 Blacklisting policies

Finally a peer in $\mathcal{N}_I(p)$ is blacklisted if one of the following criterion is met:

[4] $RTT_q(p)$ are locally cached at p so that they may be available a priori. Active measurements could also be used to quickly estimate the RTT.

Table 8.1. Number of PCs per subnet.

Subnet	1	2	3	4
Number of PCs	43	63	60	38

Table 8.2. RTTs in ms between areas of peers.

	1	2	3	4
1	20 ± 10%	80 ± 10%	120 ± 10%	160 ± 10%
2	80 ± 10%	20 ± 10%	140 ± 10%	240 ± 10%
3	120 ± 10%	170 ± 10%	20 ± 10%	200 ± 10%
4	160 ± 10%	240 ± 10%	200 ± 10%	20 ± 10%

CMR: the ratio of corrupted/late chunks among the last 100 chunks received by p from q exceeds a threshold of 5%;
PLOSS: the packet loss rate from q to p exceed a threshold of 3%; measured over the last 300 packets received;
RTT: $RTT_q(p)$ is greater than 200 ms.

Combining add and drop criteria we define 12 different overlay construction and maintenance filters. In the following, we name them stating the "ADD"-"DROP" policies, e.g., BW-RTT for add BW and drop RTT. Sect. 8.6 report results for different resulting combinations. Blacklisting can be applied (or not) to all of them, and its impact will be studied selectively. We tested also other metrics and combinations, but we only report interesting ones. RND-RND is used as a benchmark, as it is a policy based on pure sampling of the swarm.

8.5 Testbed Configuration

Following the scientific approach, we need to benchmark the different algorithms in a known and reproducible scenario. To this aim, we run experiments in a possibly complex but fully controlled network to avoid fluctuation and randomness due to external impairment. The test-bed is built in labs available at Politecnico di Torino with 204 PCs divided in four different subnets. Table 8.1 shows the number of PCs available in each subnet. We used `tc`, the standard Linux Traffic Controller tool, together with the netem option to setup RTTs among subnets and packet dropping probability when needed. The RTT distribution is described in Table 8.2. The upload bandwidth is limited by the application itself, exploiting the feature of a simple leaky bucket (its memory being 10MB) to limit the application data rate to a given desired value. Peer upload capacities C_p are shown in Table 8.3. Each

Table 8.3. Characteristics of peer classes.

Class	Bandwidth	Percentage of Peers
1	5 Mb/s ± 10%	10 %
2	1.6 Mb/s ± 10%	35 %
3	0.64 Mb/s ± 10%	35 %
4	0.2 Mb/s, ± 10%	20 %

PC is running 5 peers (5 independent instances of PeerStreamer) simultaneously, thus, a swarm of 1020 peers is considered in every experiment. The source peer run in an independent server (not belonging to any of the subnets). It injects in the swarm 5 copies of each newly generated chunk, corresponding to roughly 6 Mbit/s (see below).

The well known *Pink of the Aerosmith* video sequence has been used as benchmark. The nominal sequence length corresponds to 200s, its spatial resolution is 352×240 pixels, while the time resolution is 25 frame/s. The sequence is looped for a total stream duration of about 15 min. After the initial 12 min of experiment, each peer starts saving on local disk a 3 min long video that we use to compute QoE metrics.

We selected the H.264/AVC codec for encoding the video sequence. A hierarchical type-B frames prediction scheme has been used, obtaining 4 different kinds of frames that, in order of importance, are: IDR, P, B and b. The GOP structure is IDR×8 {P,B,b,b}. The nominal video rate of the encoder r_s is 1.2 Mb/s if not otherwise specified. This corresponds to a system load $\rho = 0.9$ – defined as $\rho = r_s/E[C_p]$ where $E[C_p] = 1.324$ Mbit/s is the average upload bandwidth of peers.

The source node generates a new chunk at regular time, i.e., every new frame, also enabling a stricter real-time streaming. The chunk size is instead highly variable. Each peer implements a chunk buffer of 150 chunks. Given the one-frame⇔one-chunk mapping, and 25 fps of the video, this corresponds to a buffer of 6s, i.e., the playout deadline is only 6s.

8.5.1 Network scenarios

The general networking scenario sketched above is declined in three different flavors that allow the exploration of different significant situations. The first scenario, *G_Homo* hereafter, is geographically homogeneous: the distribution of the peers of different C_p classes is the same in any areas, so that there is the same distribution of bandwidth everywhere. This scenario is useful to understand the fundamental behavior of different neighborhood filtering strategies.

The second scenario, *G_Bias* hereafter, assumes that bandwidth rich peers (class

1) are all concentrated in a single subnet. This situation is particularly challenging for a topology management system that tries to localize traffic and reduce the network footprint of the application.

The third and final scenario, *Lossy* hereafter, is again geographically homogeneous, but the long-haul connections between the subnets 1–3, 1–4, 2–3, 2–4 are subject to packet loss with probability $p = 0.05$, while only the intra-subnet links and the links between 1–2 and 3–4 are lossless. This situation is particularly useful to understand if black-listing can really help in building better topologies, or if its use should be limited to isolate misbehaving and malicious nodes.

8.6 Performance Evaluation

As performance indices to assess the QoE, for each peer p, we consider the *frame loss probability*, $F_{loss}(p)$, and the SSIM (Structural Similarity Index) $S_{ssim}(p)$, a well-known method for measuring the similarity between two images in the multimedia field [20]. Given the highly structured organization of the video streams, the degradation of the received video quality become typically noticeable for values of $F_{loss}(p)$ higher than 1%, while loss probability of few percent (3-4%) significantly impair the QoE. In the following, we report both average frame loss, $F_{loss} = E_p[F_{loss}(p)]$, and the percentage of peers that suffer $F_{loss}(p)$ larger than 1% and 3% respectively.

Performance however should also take into account the cost for the network to support the application. As *network cost* ζ we consider the average of the distance traveled by information units. Formally, let $b_q(p)$ the number of bits peer p received from peer q; the peer p network cost $\zeta(p)$ is computed as

$$\zeta(p) = \frac{\sum_q RTT_q(p) b_q(p)}{\sum_q b_q(p)} \tag{8.2}$$

while the average network cost is $\zeta = E_p[\zeta(p)]$

8.6.1 *G_Homo* scenario

We start considering the case in which the distribution of C_p is geographically homogeneous.

The top plot in Fig. 8.2 shows the average frame loss probability experienced by different policies, while center and bottom plots report, respectively, the percentages of peers that experienced $F_{loss}(p) > 0.01$ and $F_{loss}(p) > 0.03$.

RND-RND is the reference, and we immediately observe that the other algorithms modify the loss *distribution*, i.e., they can have a different impact on different percentiles. For instance BW-RTT improves the average loss rate and the percentage of peers with $F_{loss}(p) > 0.01$, but at the expense of the percentage of peers with

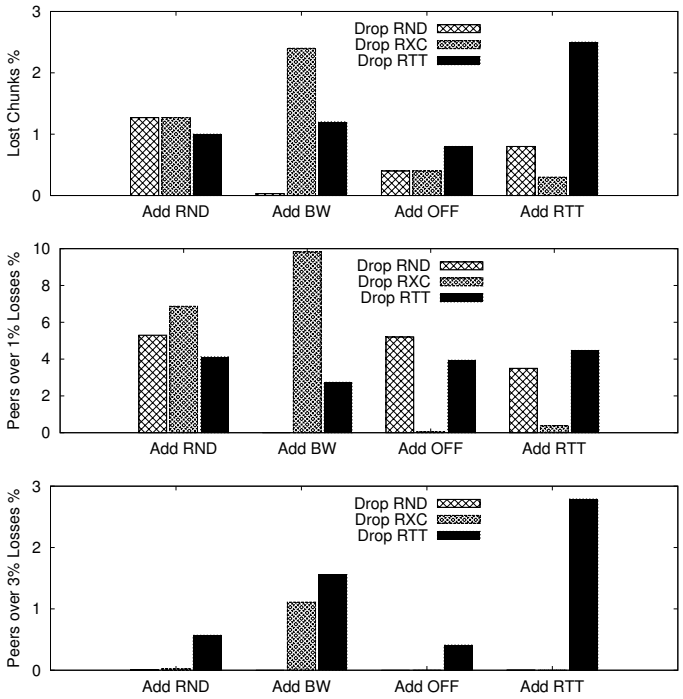

Figure 8.2. Frame loss for different strategies in *G_Homo* scenario: F_{loss} (average) (top), percentage of peers whose $Floss(p) > 0.01$ (center), percentage of peers whose $Floss(p) > 0.03$ (bottom).

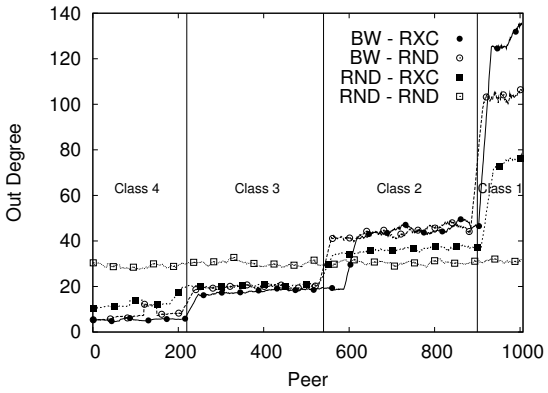

Figure 8.3. Out-degree distribution of peers, *G_Homo* scenario.

bad quality ($F_{loss}(p) > 0.03$), while RTT-RTT improves the number of peers with $F_{loss}(p) > 0.01$, but both the average and the percentage of peers with bad quality ($F_{loss}(p) > 0.03$) are worse.

In general adding policies sensitive to peers bandwidth (BW and OFF for adding

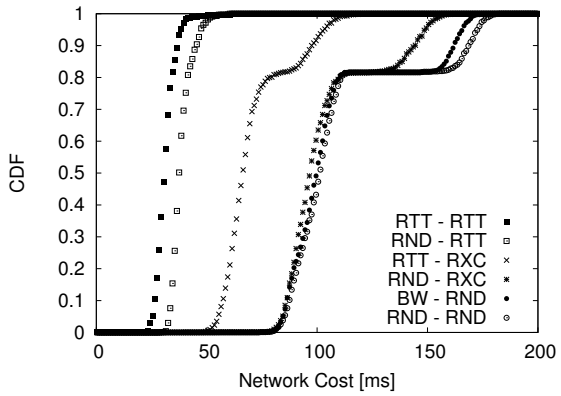

Figure 8.4. CDF of the distance traveled by information units, *G_Homo* scenario.

and RXC for dropping) appears to be the more affective in reducing the losses. However the behavior of BW-RXC for which F_{loss} tops to 2.5% indicates that using a single metric for selecting the neighborhood can be dangerous. BW-RXC biases too much the choices toward high bandwidth peers, which become congested and are not able to sustain the system demand. To better grasp these effects, Fig. 8.3 reports the smoothed[5] histogram of the out-degree $N_O(p)$. Observe that $N_O(p)$ of peers belonging to different classes is significantly different as long as bandwidth aware policies are adopted; out-degrees are instead independent for RND-RND as expected. In principle it would be desirable to have an out-degree of a peer proportional to its up-link bandwidth. This is roughly achieved by adopting BW-RND policy. Under BW-RXC, instead, the degree distribution depends too much on C_p. As a result, high bandwidth peers tends to be oversubscribed while medium and low bandwidth peers may be underutilized.

Policies sensitive to RTT perform well in the considered scenario, with the exception of RTT-RTT, which is again too aggressive in strictly selecting the closest parents. Indeed, as already observed in [36], policies that force a too strict localization of traffic may induce significant performance degradations due to poor topological properties of the swarm. To complement previous information Fig. 8.4 reports the CDF of network cost $\zeta(p)$. As expected, RTT aware policies tends to significantly reduce this index thanks to their ability to select parents within the same area.

As a first consideration, we can say that: i) bandwidth aware policies improve the application performance; ii) RTT aware policies reduce the network cost without endangering significantly the video quality; iii) the preference toward high bandwidth

[5]The distribution of $N_O(p)$ inside classes is binomial as expected from theory. This distribution results in a large noisiness of the plot, so we apply a smoothing window of length 30 in plotting, basically showing the average N_O in each class.

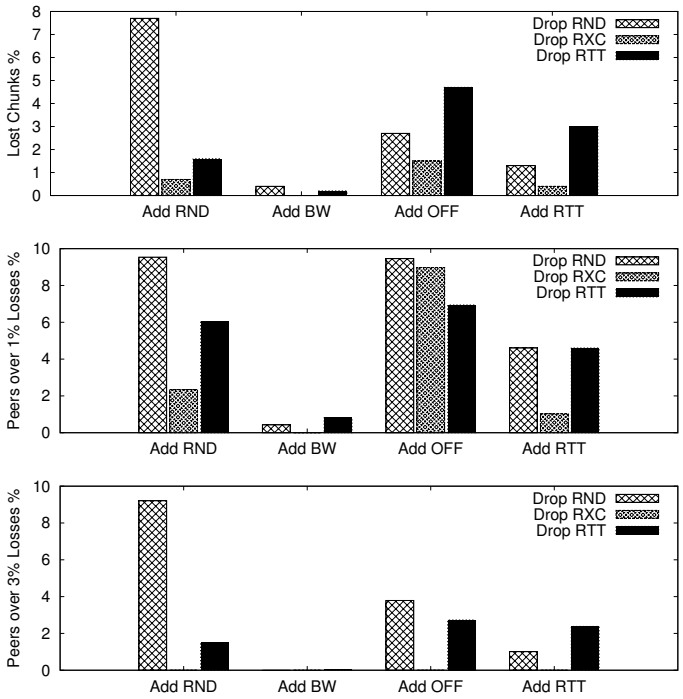

Figure 8.5. Frame loss for different strategies in *G_Homo* scenario with $N_I = 20$: F_{loss} (average) (top), percentage of peers whose $Floss(p) > 0.01$ (center), percentage of peers whose $Floss(p) > 0.03$ (bottom).

peers/nearby peers must be tempered to achieve good performance. For instance a policy like RTT-RXC improves quality and reduces the network cost at the same time.

Next, we consider the same network scenario but we set $N_I = 20$. This is a more critical situation and we expect that choosing the good parents is more important and expect RND policies to suffer more. Notice that the value of N_I is related to the total number of peers, so that for actual global broadcasting to millions of peers having a policy that performs well with a small N_I is very important, as the signaling overhead increases with N_I.

Results are plotted in Fig. 8.5 (the y-scales in Figs. 8.2 and 8.5 are different for readability reasons, and this is the reason why at first sight some policies seem to perform better with a smaller N_I). The performance of RND-RND significantly degrades in this case. The reason is that the out degree of Class 1 peers under RND-RND is often not enough to fully exploit their bandwidth. Bandwidth aware strategies, instead, successfully adapt $N_O(p)$ to C_p maintaining high performance. Also RTT-RND and RTT-RTT, which are bandwidth unaware, perform better than RND-RND, since RTT-aware selection policies reduce the latency between an offer

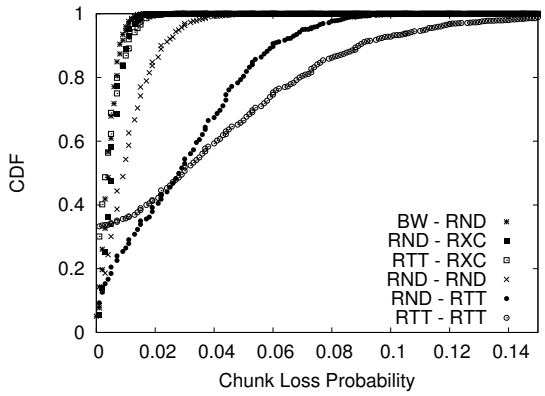

Figure 8.6. CDF of the frame loss probability for four different strategies, *G_Bias* scenario.

and the actual chunk transmission that follows it, helping in exploiting the peer's bandwidth. Results for network cost are similar to those in Fig. 8.4 and are not reported for the sake of brevity.

Random selection policies, which are widely employed by the community, are robust, but perform well only if the number of peers in the neighborhood is fairly large, which, for live streaming applications, implies a large overhead for signaling. As already seen with $N_I = 30$, also in this case the policy that combines bandwidth and RTT awarenesses (RTT-RXC) definitely improves both performance and network costs.

8.6.2 *G_Bias* scenario

Maintaining unchanged the C_p distribution, we localize all Class 1 peers in geographical area 1. This scenario, in principle, constitutes a challenge for the policies that try to localize traffic. Indeed as side effect of the localization we can potentially have a "riches with riches", "poors with poors" clusterization effect that may endanger the video quality perceived by peers in other geographical regions than 1.

Fig. 8.6 reports the CDF of $F_{loss}(p)$ for the five strategies performing better in the *G_Homo* scenario, plus the benchmark RND-RND. In this case if RTT is the only metric used as in RTT-RTT, the performance degrades unacceptably, and peers in area 1 are in practice the only one receiving a good service. However, observe that strategy RTT-RXC performs as well as RND-RXC and BW-RND, but it can also reduce the network cost, as shown in Fig. 8.7 that reports the CDF of $\zeta(p)$.

This result essentially proves that also in *G_Bias* conditions it is possible to partially localize the traffic without endangering the video quality perceived by the user, as long as RTT awareness is tempered with some bandwidth awareness.

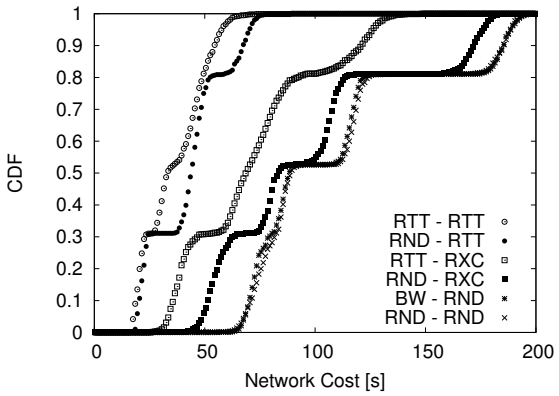

Figure 8.7. CDF of distance traveled by information units, *G_Bias*scenario.

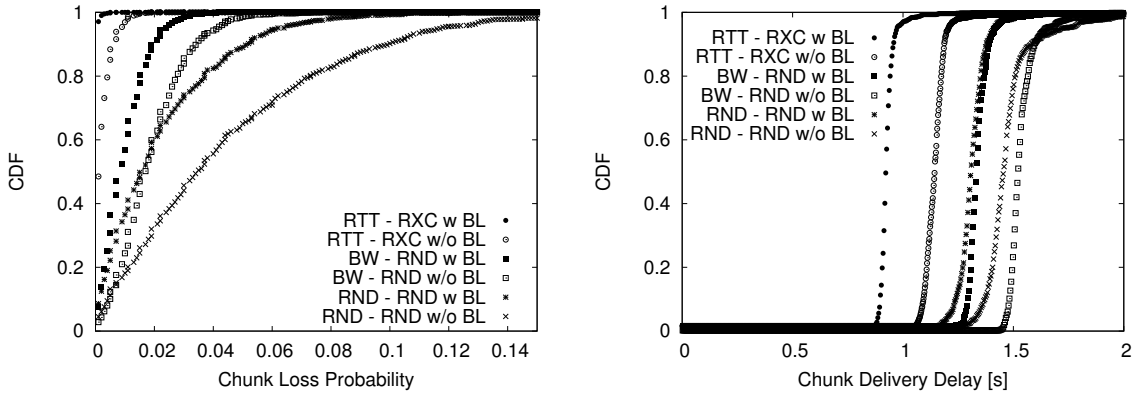

Figure 8.8. CDF of chunk loss probability (top) and CDF of chunk delivery delays (bottom) for three different strategies with and without adopting blacklist mechanism in *Lossy* scenario.

8.6.3 *Lossy* scenario

We consider another challenging scenario in which large bandwidth peers are uniformly distributed over the four subnets, but packet losses are present in some long haul connections. In this case, we expect that blacklisting can play a significant role to avoid selecting lossy paths. Indeed, exploiting the blacklist mechanism every peer should identify and abandon poorly performing parents, biasing the neighborhood toward good performing peers. This effect should reinforce policies that naturally bias the selection of neighbor peers employing peer quality. We only plot the results for RND-RND, BW-RND and RTT-RXC as these latter have emerged as the most promising criteria and RND-RND is the benchmark. The performance of other policies did not show any really interesting behavior in this scenario.

Fig 8.8 plots the CDF of frame losses (top) and the CDF of chunks delivery

Table 8.4. Average fractions of incoming traffic for Cluster 2.

	1 - good	2 - local	3 - bad	4 - bad + far
RND - RND w/o BL	0.23	0.32	0.28	0.15
RND - RND w BL	0.28	0.34	0.24	0.12
BW - RND w/o BL	0.22	0.35	0.27	0.14
BW - RND w BL	0.23	0.36	0.24	0.13
RTT - RXCH w/o BL	0.12	0.68	0.11	0.07
RTT - RXCH w BL	0.13	0.70	0.09	0.05

delays (bottom) for the selected policies. Blacklisting improves the performance of every policy. RTT-RXC emerges again as the most performing policy and with blacklisting practically all peers are able to receive all chunks. This is an excellent result, since the system is facing a very challenging scenario while working with a load of 0.9.

Benefits of the blacklisting mechanism are confirmed by Table 8.4 that reports the normalized volume of incoming traffic for peers in cluster 2 from peers in all clusters. Keeping in mind that in *Lossy* scenario peers belonging to cluster 2 experience lossy paths from/towards peers in cluster 3 and 4 (as explained in Sec. 8.5), it is easy to see that volumes of incoming traffic from cluster 3 and 4 are nicely reduced thanks to blacklisting mechanism.

8.6.4 Video performance versus load

In the previous sections we have benchmarked the system versus increasingly difficult scenarios, showing the benefits and drawbacks of overlay topology filtering strategies. Now we summarize the results by depicting the actual average QoE by reporting S_{ssim} for different policies and different system loads. We consider the final *Lossy* scenario, and we increase the video rate from 0.6 Mb/s to 1.4 Mb/s. Recall that $E[C_p] = 1.324$ Mb/s.

Fig. 8.9 shows S_{ssim} considering RND-RND, BW-RND and RTT-RXC with and without blacklisting. S_{ssim} is a measure of the distortion of the received image compared against the original source (before encoding and chunkization). It is a highly non linear metric in decimal values between -1 and 1. Negative values correspond to negative images, so are not normally considered at all. Values above 0.95 are typically considered of good quality. S_{ssim} has been computed considering video frames received by 200 peers (50 for each class), and then averaging among all of them. The initial 12 min of the video have been discarded to focus on steady state performance. The S_{ssim} is computed for 1 min of the video given the enormous computational burden of this task.

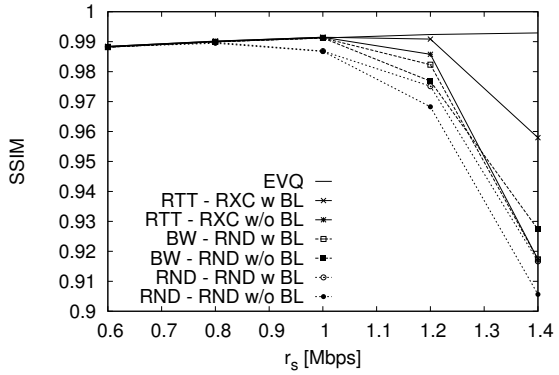

Figure 8.9. S_{ssim} index when varying video rate r_s in *Lossy* scenario.

The EVQ (Encoded Video Quality) curve in the plot is the reference value for the encoding level and it obviously increases steadily as r_s increases. When the system load $\rho < 1$, S_{ssim} increases for increasing r_s thanks to the higher quality of the encoded video. As soon as the system is overloaded, the S_{ssim} rapidly drops due to missing chunks which impair the quality of the received video. Notice how RTT-RXC scheme outperforms RND-RND and BW-RND for every value of r_s. Fig. 8.9 also shows the benefits of the blacklist mechanism for every scheme.

Chapter 9

Conclusions

The design of P2P systems so as to minimize their impact on the network has been invoked and studied in the past few years, but most studies remains theoretic or based on simulations.

In the context of P2P-TV systems, Chapter 7 presented a complete work that, starting from simple heuristic considerations, design a network-aware topology management system based on distributed measurements. The topology management system is first evaluated in simulations, so as to verify the performances in a simplified and controlled environment. Then, building on a P2P-TV system already developed within the NAPA-WINE project, a fully distributed implementation of the system named PeerStreamer, which includes the topology management as well as real video distribution, has been presented and tested over the Internet, with a mix of peers running on normal hosts and peers running in PlanetLab nodes in order to increase the size of the distribution swarms.

In Chapter 8 we evaluate the impact of P2P-TV overlay topologies through extensive benchmark campaigns in large scale test-beds. Leveraging the PeerStreamer application developed within the framework of the EU NAPA-WINE project, we have implemented a flexible algorithm which lets us drive the strategies for building neighborhoods of peers and, hence, the overall topology, without changing other sub-systems of the application, thus isolating the impact of topology management from other effects. In a controlled networking environment, we have run a large campaign of experiments measuring the impact of different filtering functions applied to the management of peer neighborhoods. Confirming considerations presented in Chapter 7, Chapter 8 shows that proper management, based on simple RTT measurements to add peers, coupled with an estimation of the quality of the peer-to-peer relation to drop them, leads to an optimal situation where the performance of the application is improved while the network usage is reduced.

In our evaluation the P2P-TV system performance we considered not only network level metrics, but also the quality of experience (QoE) of users based on the

reconstruction of the average PSNR or SSIM of the video itself at playback. This procedure enabled us to verify that a topology built following a network-aware, localization-oriented metric can not only reduce the network costs, but also increase the QoE of users, leading to a win-win situation that can be the driver for the adoption of such a technology.

9.1 Future Directions

Given the good the results obtained in a controlled environment and presented in Chapter 8, we aim at replicating the same experiments in an Internet-wide scenario such as the one offered by PlanetLab system. Confirming in such scenario the benefits of network-aware policies would be an excellent result.

Appendix A

Acronyms

ACK acknowledgement

ASP Adaptive Signalling Protocol

CDF cumulative distribution function

DVB-T Digital Video Broadcasting - Terrestrial

EVQ encoded video quality

FIFO first in first out

GOP group of pictures

HD-TV high definition signal

HRC Hose Rate Control

IDR instantaneous decoder refresh

LEDBAT Low Extra Delay Background Transport

MPEG TS Moving Picture Experts Group Transport Stream

MSE Mean Squared Error

NAPA-WINE Network-Aware P2P-TV Application over Wise Networks

NTP Network Time Protocol

P2P peer-to-peer

P2P-TV peer-to-peer streaming

PSNR Peak Signal-to-Noise Ratio

QoE quality of experience

RTT round trip time

SSIM Structural Similarity Index

STREP Specific Targeted Research Projects

TCP Transmission Control Protocol

UDP User Datagram Protocol

VoIP Voice over Internet Protocol

Bibliography

[1] "Meridian, http://www.cs.cornell.edu/people/egs/meridian/."

[2] "PPLive, http://www.pptv.com/en/."

[3] "CoolStreaming, http://www.coolstreaming.us/."

[4] "SopCast, http://www.sopcast.org/."

[5] L. Massoulié, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized decentralized broadcasting algorithms," in *IEEE INFOCOM*, Anchorage, AK, May 2007.

[6] S. Sanghavi, B. Hajek, and L. Massoulié, "Gossiping with multiple messages," in *IEEE INFOCOM*, Anchorage, AK, May 2007.

[7] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg, "Epidemic live streaming: optimal performance trade-offs." in *SIGMETRICS*, Annapolis, MD, June 2008.

[8] 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 *P2PMMS*, Singapore, November 2005.

[9] Y. Liu, "On the minimum delay peer-to-peer video streaming: how realtime can it be?" in *ACM Multimedia*, Augsburg, DE, September 2007.

[10] D. Ciullo, M. A. Garcia, A. Horvath, E. Leonardi, M. Mellia, D. Rossi, M. Telek, and P. Veglia, "Network awareness of P2P live streaming applications: a measurement study," *IEEE Transanctions on Multimedia*, vol. 12, no. 1, pp. 54–63, January 2010.

[11] X. Zhang, J. Liu, and T. Yum, "Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming," in *IEEE INFOCOM*, Miami, FL, March 2005.

[12] F. Picconi and L. Massoulié, "Is there a future for mesh-based live video streaming?" in *IEEE P2P*, Aachen, DE, September 2008.

[13] "NAPA-WINE, http://www.napa-wine.eu," 2008-2011.

[14] "PeerStreamer, http://peerstreamer.org."

[15] A. C. da Silva, E. Leonardi, M. Mellia, and M. Meo, "A bandwidth-aware scheduling strategy for P2P-TV systems," in *IEEE P2P*, Aachen, DE, September 2008.

[16] T. Small, B. Liang, and B. Li, "Scaling laws and tradeoffs in Peer-to-Peer live multimedia streaming," in *ACM Multimedia*, Santa Barbara, CA, October 2006.

[17] R. Lobb, A. P. Couto da Silva, E. Leonardi, M. Mellia, and M. Meo, "Adaptive overlay topology for mesh-based p2p-tv systems," in *ACM NOSSDAV*, Williamsburg, Virginia, June 2009.

[18] "P2PTV-Sim, http://www.napa-wine.eu/cgi-bin/twiki/view/public/p2ptvsim."

[19] S. Shalunov and G. Hazel, "Low Extra Delay Background Transport (LEDBAT)," IETF, Internet Draft draft-ietf-ledbat-congestion-02, July 2010.

[20] Z. Wang, A. C. Bovik2, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[21] A. Couto da Silva, E. Leonardi, M. Mellia, and M. Meo, "Exploiting Heterogeneity in P2P Video Streaming," *IEEE Transactions on Computers*, December 2010.

[22] E. Setton, J. Noh, and B. Girod, "Low latency video streaming over peer-to-peer networks," in *IEEE ICME*, Toronto, CA, July 2006.

[23] Z. Liu, Y. Shen, K. W. Ross, S. S. Panwar, and Y. Wang, "Layerp2p: using layered video chunks in p2p live streaming," *IEEE Transaction on Multimedia.*, vol. 11, no. 7, pp. 1340–1352, 2009.

[24] H. Xie, R. Y. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz, "P4P: provider portal for applications," in *SIGCOMM*, Seattle, WA, August 2008.

[25] Z. Shen and R. Zimmermann, "Isp-friendly peer selection in p2p networks," in *ACM Multimedia*, Beijing, China, October 2009.

[26] C.-H. Lai, Y.-W. Chan, and Y.-C. Chung, "A construction of peer-to-peer streaming system based on flexible locality-aware overlay networks," in *Advances in Grid and Pervasive Computing*, ser. Lecture Notes in Computer Science, S. Wu, L. Yang, and T. Xu, Eds. Springer Berlin / Heidelberg, 2008, vol. 5036, pp. 296–307. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68083-3_30

[27] X. Cheng, F. Wang, J. Liu, and K. Xu, "Collaborative delay-aware scheduling in peer-to-peer ugc video sharing," in *Proceedings of the 20th international workshop on Network and operating systems support for digital audio and video*, ser. NOSSDAV '10. New York, NY, USA: ACM, 2010, pp. 105–110. [Online]. Available: http://doi.acm.org/10.1145/1806565.1806591

[28] R. Fortuna, E. Leonardi, M. Mellia, M. Meo, and S. Traverso, "Qoe in pull based p2p-tv systems: Overlay topology design tradeoffs," in *IEEE P2P 2010*, Aug. 2010, pp. 1–10.

[29] L. Abeni, C. Kiraly, and R. Lo Cigno, "On the optimal scheduling of streaming applications in unstructured meshes," in *IFIP Networking*, Aachen, DE, May 2009.

[30] ——, "Scheduling P2P multimedia streams: Can we achieve performance and robustness?" in *3-rd IEEE International Conference on Internet Multimedia Systems Architecture and Applications (IMSAA-09)*, 2009, pp. 1–6.

[31] A. Russo and R. Lo Cigno, "Delay-Aware Push/Pull Protocols for Live Video Streaming in P2P Systems," in *IEEE International Conference on Communications (ICC'10)*, May 2010, pp. 1–5.

[32] C. Liang, Y. Guo, and Y. Liu, "Investigating the scheduling sensitivity of P2P video streaming: An experimental study," *IEEE Transactions on Multimedia*, vol. 11, no. 3, pp. 348–360, april 2009.

[33] D. Ren, Y. T. H. Li, and S. H. G. Chan, "On reducing mesh delay for peer-to-peer live streaming," in *IEEE INFOCOM*, Phoenix, Arizona, April 2008.

[34] T. Locher, R. Meier, S. Schmid, and R. Wattenhofer, "Push-to-pull peer-to-peer live streaming," in *Lecture notes in computer science*, Berlin, DE, 2007.

[35] J. Chakareski, "Topology construction and resource allocation in p2p live streaming," in *Intelligent Multimedia Communication: Techniques and Applications*, ser. Studies in Computational Intelligence, C. Chen, Z. Li, and S. Lian, Eds. Springer Berlin / Heidelberg, 2010, vol. 280, pp. 217–251.

[36] A. Couto da Silva, E. Leonardi, M. Mellia, and M. Meo, "Chunk Distribution in Mesh-Based Large Scale P2P Streaming Systems: a Fluid Approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 3, pp. 451–463, March 2011.

[37] J. Seibert, D. Zage, F. S., Nita-rotaru, and C., "Experimental comparison of peer-to-peer streaming overlays: An application perspective," in *33rd IEEE Conference on Local Computer Networks, Proceedings*, 2008, pp. 1–6.

[38] J. Douceur, J. Mickens, T. Moscibroda, and D. Panigrahi, "Collaborative measurements of upload speeds in p2p systems," in *IEEE INFOCOM*, San Diego, CA, March 2010.

[39] J. Seedorf, S. Kiesel, and M. Stiemerling, "Traffic localization for p2p-applications: The alto approach," in *IEEE P2P*, Seattle, WA, September 2009.

[40] L. Abeni, C. Kiraly, and R. Lo Cigno, "Robust scheduling of video streams in network-aware p2p applications," in *IEEE ICC 2010*, May 2010, pp. 1–5.

[41] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in IPTPS, Ithaca, NY, February 2005.

[42] E. Setton, J. Noh, and B. Girod, "Congestion-distortion optimized peer-to-peer video streaming," in *IEEE ICIP*, Atlanta, Georgia, October 2006.

[43] L. Abeni, C. Kiraly, A. Russo, M. Biazzini, and R. Lo Cigno, "Design and implementation of a generic library for P2P streaming," in *Workshop on Advanced Video Streaming Techniques for Peer-to-Peer Networks and Social Networking, 2010*, Florence, Italy, October 2010, pp. 1–6.

[44] N. Tölgyesi and M. Jelasity, "Adaptive peer sampling with newscast," in *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, ser. Euro-Par '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 523–534.

[45] X. Jiang, Y. Dong, D. Xu, and B. Bhargava, "Gnustream: a P2P media streaming system prototype," in *Proc. of the 2003 International Conf. on Multimedia and Expo - Volume 1*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 325–328.

[46] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "Promise: peer-to-peer media streaming using collectcast," in *Proc. of ACM MULTIMEDIA '03*. New York, NY, USA: ACM, 2003, pp. 45–54.

[47] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, pp. 149–160, August 2001.

[48] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang, "Inside the new coolstreaming: Principles, measurements and performance implications," in *IEEE INFOCOM*, 2008, pp. 1031–1039.

[49] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *IEEE INFOCOM*, 2006.

[50] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," in *INFOCOM 2007. 26th IEEE International Conf. on Computer Communications. IEEE*, may 2007, pp. 1415 –1423.

[51] Y. Liu, "On the minimum delay peer-to-peer video streaming: how realtime can it be?" in *ACM Multimedia*, Augsburg, DE, September 2007.