Live Video and IP-TV

Maria Luisa Merani and Daniela Saladino

Abstract P2P architectures designed for video broadcasting have in the very last years gained a prominent role. This Chapter aims at providing a comprehensive insight of the most recent advances in the field, focusing on live video streaming. The first part of the Chapter puts forth a classification of P2P video solutions, adopting alternative sorting criteria that hide different design approaches. It then concentrates on the conceptually attractive issue of data diffusion process within the P2P overlay. An overview of the most interesting P2P IP-TV systems currently available over the Internet is also provided, and the most salient features they exhibit highlighted. Next, the definition of the quality of experience (QoE) for a system user, as well as the recording of the whole system performance via local and centralized measurement approaches, is commented upon. The second part of the Chapter completes the view, bringing up the modeling efforts that capture the main characteristics and limits of a P2P streaming system, both analytically and numerically. The Chapter is closed by a pristine look at some challenging, open questions: the issue of peers that lie behind NAT and firewalls is discussed; the benefits and the limits of cross-layer design are commented, with a specific emphasis onto the adoption of advanced coding techniques.

Maria Luisa Merani
Department of Information Engineering, University of Modena and Reggio Emilia, Italy, e-mail: marialuisa.merani@unimore.it

Daniela Saladino
Department of Information Engineering, University of Modena and Reggio Emilia, Italy, e-mail: daniela.saladino@unimore.it

1 Live streaming and IP-TV

Traditionally, video delivery over the Internet relies upon the consolidated client-server paradigm. In this perspective a centralized server has to be accessed by all clients that wish to download the desired video stream, as Fig.1 illustrates.

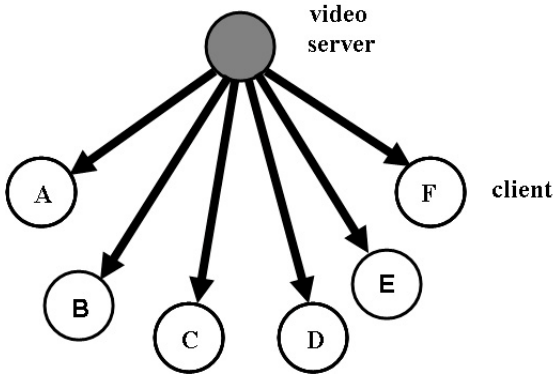

Fig. 1 The conventional client-server architecture for video delivery over the Internet

This simplified view immediately suggests that the server access bandwidth is the most limiting factor against system scalability, at least in terms of network resources. Referring to the case of constant bit rate connections, when the number of concurrently active clients increases and the sum of the bandwidths that their flows require equals the server access bandwidth, then the video streaming system saturates. No more users can be supported, otherwise congestion will soon appear and markedly penalize the throughput of the video application.

More generally, the client-server approach entails that the video server be the edge of as many unicast connections as the number of clients: one video stream per client is individually and separately taken to destination, consuming bandwidth and network resources, and possibly generating congestion along all traversed paths. Definitely, not a smart solution for several video applications: it suffices to think of the broadcasting of television events, where the same information has to be simultaneously delivered to each subscriber.

Multicasting at the IP-layer, probably the cleanest solution from a conceptual viewpoint, was first proposed to relieve the problem, but IP multicast never took place over the global Internet [1]. The violation of the stateless principle of IP routers, the lack of scalability, the increased difficulty in performing congestion and error control at transport layer on multicast connections were probably the technical factors that mostly limited its inception.

One of the novelty of P2P for video delivery over the Internet resides in moving the multicast approach to the application layer. Another unique point being that it is up to the same end-users of the multicast group to collaborate in the process of

swarming the information across the network to other users: a node in the overlay will not only receive the desired video, but will also cooperate to distribute the video to other peers. As we will see in next Section, the way the video information propagates across the peer overlay provides the most relevant metric to classify P2P systems for video delivery.

In P2P solutions the video stream is divided into relatively small, equal size chunks, that typically contain a few seconds of the original video. Participating peers make some of their untapped resources available to the system, most notably their upload bandwidth, to pass the video chunks they already own to other peers: this greatly relieves the burden on the original streaming server. Moreover, the infrastructure requirements of the application-layer approach are so minimal, that they really make P2P an attractive candidate for video distribution over the Internet.

Undoubtedly, the low cost of such application-layer approach makes it a strong candidate to satisfy the demand for video distribution over the Internet, for a variety of heterogeneous services.

To conclude this introductory discussion, observe that P2P had already gained its slice of popularity in file sharing applications, well before it was extended to video delivery. It is however worth underlining that the migration of the P2P approach to this new realm was not painless: video applications exhibit peculiar features, unknown to file sharing, most prominently the real-time constraints that the great majority of them imposes on information delivery; moreover, video services are and will be bandwidth eager, by far a more challenging characteristics than VoIP, another very common real-time application.

Hence, the focus shifts: whereas efficient indexing and searching techniques are of paramount importance in P2P systems for file sharing, a careful scheduling to minimize delays is required when the P2P overlay handles video, as well as a satisfying resilience to peer churning, i.e., to a swift increase/decrease in the number of peers within the overlay. The ultimate goal is to warrant a satisfying Quality of Experience (QoE) to the peers viewing the video, as well as smooth, confined quality variations.

2 A taxonomy of P2P video broadcasting systems

Video applications over the Internet span quite a broadcast range: from video-conferencing, imposing very strict time constraints, to live video streaming with nearly synchronized users – this is what we will mainly refer to in what follows, interchangeably using the IP-TV and video broadcasting terms –, to video on demand, the most delay-tolerant category.

Let us therefore illustrate the main classification of the P2P systems devised for IP-TV, employing the approach to overlay construction as the sorting criterion. P2P solutions are accordingly distinguished in tree-based and mesh-based architectures [2]. In the former the video stream propagates from the source via a tree of peers, so that gradually the video spreads over the entire overlay, from the root of the tree to

its last leaves. Recall that the tree is built at the application layer, and relies on the underlying unicast IP connections.

Fig.2 illustrates the idea: in this example, peers A and B receive the video from the source root and then forward it to peers C and D, E and F, respectively; in turn, peer C is in charge of delivering it to leaf peers G and H, peer F to leaf peer I.

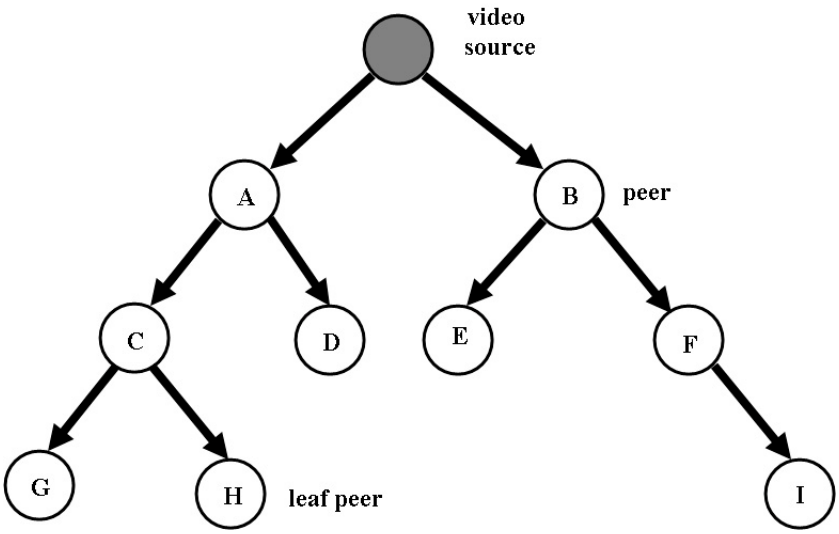

Fig. 2 High level scheme for a tree based P2P overlay

This solution is also termed push-based, as the video is pushed along the tree, a topology that naturally meets the multicasting demand. How is the tree formed? When a new peer has to join it, bandwidth and delay are the indicators that typically drive the choice of the parent node: the peer can select the parent depending on the round trip time from it, on the application throughput the parent experiences, on its uploading bandwidth. Avoiding loops is the constraint to respect.

Easy as it appears, this solution unfortunately exhibits several drawbacks: it is prone to outages, as the departure of a non-leaf peer from the overlay deprives all its descendants of the video content; it does not utilize the upload bandwidth of leaf peers, that only receive content, but do not collaborate in distributing the video across the overlay.

Next natural step is therefore to resort to a multitree overlay, where robustness and better efficiency is achieved via multiple, disjoint trees, where peers that are leaves in a tree are not so in a different one. Fig.3 reports a simple example of a multitree topology, where the differently shaded arrows identify different trees. In the multitree topology, the source encodes the video stream into multiple substreams, each substream flowing on a different tree. A peer typically joins more trees, depending on its access link bandwidth, and experiences a quality that depends on the number of substreams it receives. The push mechanism of the single tree topology

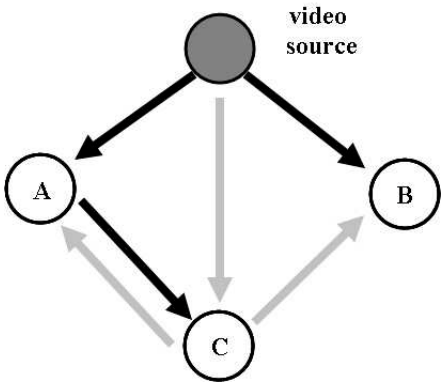

Fig. 3 The multitree concept

is retained, as packets belonging to one substream are simply forwarded from the parent node to its children peers along the corresponding tree.

The main objective to pursue in this articulated topology is to build – and maintain – short and balanced trees. It follows that peers often need to be dynamically reallocated within trees, due to the events of peer joining and departing.

The approach is completely different in mesh-based overlays, also termed pull-based architectures. There is no predefined topology here, no *a priori* notion of trees for data delivery. Rather, each peer maintains a list of partners and periodically exchanges with them information about the available data. It then "pulls" the desired blocks of video from one of the peers that advertises them, also supplying available data to other partners. Partnerships are updated at a proper rate, to ensure both robustness to failures and efficiency in the data diffusion process.

At first sight, the swarming process of this solution closely resembles popular P2P systems for file sharing like BitTorrent. There is however a significant difference: video has to obey strict time requirements, its blocks need to be delivered without suffering an unbearable delay and jitter; if it were not so, quality would be unfavorably affected. It follows that the scheduling peers adopt to pull the blocks from their parents is significantly different from the ones implemented in P2P file sharing architectures: it has to minimize delay, so as to guarantee that the majority of the downloaded chunks respect the playback deadline.

Some among the currently most popular P2P systems fall within the mesh category, and we have chosen to describe their main features in greater detail in next Section.

What is the best solution between mesh and tree? We anticipate here that both architectures enjoy benefits and drawbacks: the pull-based overlay, the most diffused in commercial systems, is simple to implement and to maintain; it is efficient, as data forwarding is not restricted to specific directions; it is resilient to swift peer dynamics [3]. Yet, it often suffers significant delays at start-up time and when switching channel, as well as non negligible time lags between peers viewing the same video.

The time spent by the peer in the process of information exchange, required to know which partner owns which information, is responsible for a large fraction of the service delay the peer suffers [4]. On the contrary, delays are definitely confined in tree-based systems, but the signaling overhead and the complexity in system design and management, to guarantee stable and resilient trees, is not to be underestimated [2].

So far, we have assumed that both tree and mesh-based overlays be organized in a flat, non hierarchical manner. Recently however, an architecture has been proposed, where peers are structured in tiers [5]. The starting point of the proposal is that in a mesh-based overlay not only most of the data blocks propagate via tree structures, but also the majority of them is delivered via an implicit stable backbone [5].

The solutions the authors put forth is therefore a layered architecture, where a first backbone tier of more stable peers with sufficient access bandwidth is organized in a tree structure, and serves more fluctuating nodes, that are placed in a second tier; this second tier can accommodate diverse overlay structures, i.e., either mesh or additional trees.

It therefore appears useful to further distinguish P2P system topologies in flat and layered categories. This classification also turns out convenient when discussing hybrid architectures, that combine the adoption of the P2P paradigm and of content replication servers, strategically placed over the Internet: these systems can well be framed within the layered category, provided the nodes of the tier-1 backbone represent the stable, always available servers, as opposed to the less predictable end-user peers of the second tier.

Last, P2P architectures for IP-TV can be distinguished on the basis of the number of their potential users, classifying P2P systems in small or large size overlays. P2P architectures serving the needs of prosumers (producers and consumers), wishing to broadcast their own video content fall within the first class, and are separated from large P2P systems tailored to the requirements of major TV broadcasters and service content providers. Within the first category, pure P2P overlays, that do not rely upon the presence of content delivery servers, represent an appealing solution; on the contrary, hybrid systems represent the most popular proposal when scalability, as well as reliable service provisioning, are the major constraints.

3 The diffusion process in mesh architectures and a reference system

As promised earlier, we now concentrate on the main characteristics exhibited by an unlayered, mesh-based overlay. The focus is often on CoolStreaming, [3], [6], [7], one of the most popular pull-based systems (at least in its original version), generally referenced in the scientific community as the benchmark.

Its developers initially preferred to describe its design as a “data-driven” approach, rather than mesh-based: indeed, no specific overlay structure confines the data flow direction.

The concept behind a mesh system is simple: peers help each other and cooperate in the delivery process, exchanging video chunks. For a generic pull-based architecture it can be affirmed that, when a new peer joins the system because it desires to view the video, the peer first contacts the origin node, where the original video is available. In the simplest case, what happens next is that the origin server redirects the new peer to a tracker, maintaining a membership list: the tracker provides the peer a set of potential partners, that the new node contacts to establish the relationships required to start receiving video chunks. As soon as such relationships are set, the peer starts receiving the buffer maps of its partners, i.e., short control messages that every peer periodically forwards to indicate its available chunks. The new peer will in turn forward its buffer maps, although at the very beginning they will reveal that the node has no content to share with other participants of the overlay. Form the buffer maps it receives, the new peer can identify its potential *parents*, i.e., the most appropriate partners from which to start to fetch the video chunks via a proper scheduling algorithm.

Note that there is a signification distinction between the terms overlay members, partners and parents [7]: the first term identifies all end-users in the overlay, wishing to view the same video; the second term refers to the peers that exchange information with the reference peer about chunk availability via their buffer maps; the third term indicates what peers (the parents) are actually providing video content to the peer (the child). The mutual relation between overlay members, partners and parents is graphically represented in Fig.4.

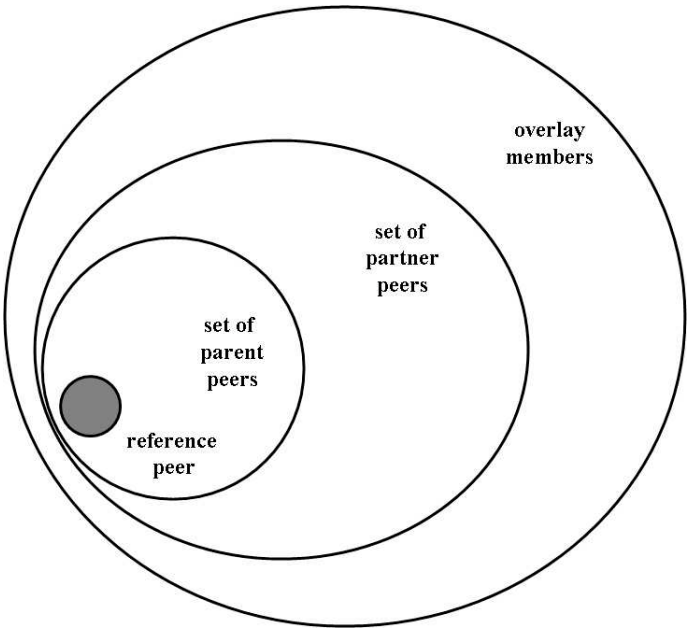

Fig. 4 Overlay members, partners and parents

As for the buffer map, its simplest structure hosts a string of zero's and one's, where the one indicates that the chunk is available at the peer, whereas the zero tells that the peer does not own the chunk; a proper offset allows to identify the first chunk of the sequence that the map refers to. This is exactly the description of the buffer maps in the original CoolStreaming prototype, that employed 120 bit long strings. Given a chunk contains one second of video, we immediately conclude that there are a couple of minutes available in the peer buffer, a typical order of magnitude for most P2P systems.

After receiving the buffer maps, the peer can request the chunks that it is missing and that are advertised by other peers. The request scheduling is a delicate issue: video chunks have to meet severe playback deadlines; if they do not, their late arrival translates into a loss, and ultimately in a degraded quality experienced by the end-user viewing the video. This constraint translates in rejecting the classical round robin scheduler, in favor of heuristics that keep the number of late or missing chunks to a minimum, possibly equal to zero.

As an example, we refer to the scheduling algorithm of the first CoolStreaming release [3], that starts by determining all potential suppliers of every chunk. The chunks with fewer suppliers are considered first, and for each chunk, a unique supplier is identified: it is the one with the highest bandwidth and enough available time to transmit the chunk. Once a schedule is specified, an individual bit sequence resembling the buffer map is sent to each supplier, indicating the chunks that the peer intends to pull from it. The video chunks are then delivered to the requesting peer via UDP connections, properly enhanced by the congestion control mechanism TFRC implements [8], [9]. In passing by, we mention that TCP is alternatively adopted by some P2P streaming systems for transporting video data, despite its overhead in terms of connection opening and closing phase, as well as retransmission handling.

Let us now more thoroughly dissect the software architecture of a peer, to logically frame the different modules that constitute it: following the description in [4], in the peer we distinguish the P2P streaming engine and the media player, as Fig.5 indicates.

The streaming engine has the responsibility of fetching video chunks from partners; of storing the retrieved chunks in a buffer; of passing the chunks to the media player. It is also in charge of providing the available chunks to those peers requesting them and to manage the buffer maps. Finally, it continuously updates the partnership list. Referring again to the first CoolStreaming, each node periodically establishes new partner relations with randomly chosen partners: the node that provides the lowest average number of chunks per unit time is discarded and replaced by a new, better performing partner.

An additional, last point deserves a further refinement in our discussion: the overlay membership management. In small to medium sized overlays, peers retrieve membership information directly from the tracker server, the unique repository of the system view: this is exactly what we have assumed in our first discussion. It is not so for large size P2P streaming systems, such as CoolStreaming: here a new node joining the system contacts the source, that redirects it to another peer, called deputy peer (rather than to the tracker server), randomly selected from its own mem-

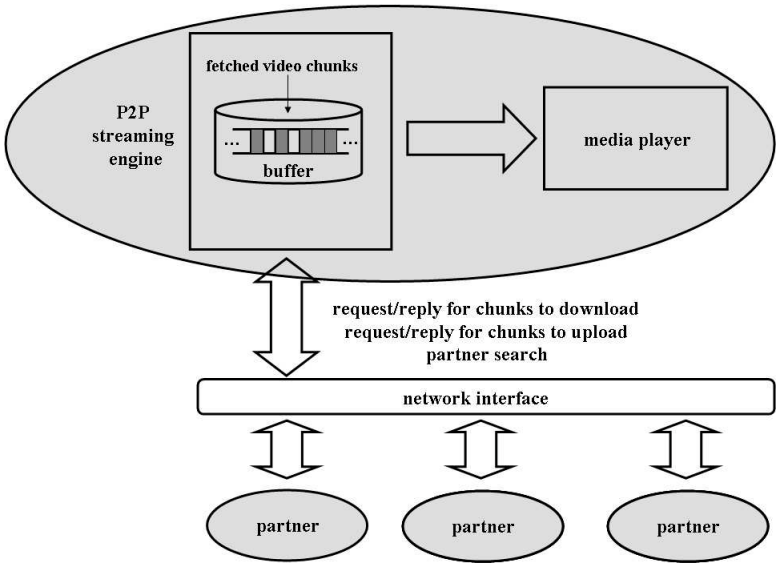

Fig. 5 Software components in a peer

bership cache. The deputy (i) provides a list of eligible partners, that the new peer contacts to establish its partner relationships, and also (ii) updates its own cache to record the entry of the new peer. This redirection guarantees a more uniform partner selection, as well as a reduced load over the video source. What is interesting to observe is that in each peer of the overlay there has to be not only a partnership cache, but also a membership cache.

Why is this cache present in all peers and how is it managed?

The answer to the first question is important: the ultimate goal is to disseminate among all peers a uniform, although forcedly partial view of the overlay members. This is necessary for the deputy functions we cited above, but also because each peer periodically consults its membership cache to replace partners, either when some of them depart or when some better performing nodes become available. This happens in a decentralized manner, without placing any burden on the origin server.

The answer to the second question is that a gossiping protocol is employed to create and update the membership cache, which in turn triggers the explanation of what we mean by the term gossip-like algorithm. A gossip algorithm, also tagged as epidemic, presents the following characteristics: a peer sends a new message (in our case it periodically announces it exists) to a random subset of other peers; in the next round these peers propagate the message in the same manner, and so do next peers that receive it. Gradually, in a totally distributed manner, the information the peer exits propagates in the overlay, contributing to the construction of the local view of the overlay members at each peer that receives it.

When discussing the weaknesses of mesh-based systems, we already evidenced that excessive initial delays are expected to plague this architecture: indeed, Cool-

Streaming earlier release had Achille's heel of long initial delays, as well as a high failure rate in joining a channel during a peer churn [7].

The diffusion process it adopted was then radically modified, and we intentionally describe it here, with the intent to understand how CoolStreaming and in general P2P streaming overlays have to evolve to fulfill commercial system requirements.

The New CoolStreaming adopts a multiple substream solution to better swarm the video among its peers; in turn, this calls for a fundamental change in the architecture of the peer buffer, in its management, as well as in the scheduling scheme; moreover, a new, hybrid push-pull mechanism that the peers adopt to download video chunks is developed. Last but not least, CoolStreaming now employs multiple servers, strategically located. Equivalently, the system does not rely on a pure P2P overlay any longer; rather, it now adopts a hybrid architecture, where the streaming capacity is proportionally amplified with the number of servers, and the content is taken closer to the end-users.

Let us see in detail the multiple substream solution and the novel buffer organization at the peer. As before, the video is divided in blocks of equal size; the novelty relies in splitting the blocks in different substreams, as Fig.6 reports. Video chunks are grouped according to the following pattern: given that K substreams have to be formed, the j-th substream ($j = 1, 2, \ldots, K$) is made of the chunks with the following sequence number in the original stream: $j + i * K$, $i = 0, 1, 2, \ldots$.

A peer can subscribe to one or more substreams, fetching the corresponding chunks from multiple parents. Although no specific coding technique is employed, the critical point resides in maintaining the synchronization among different substreams.

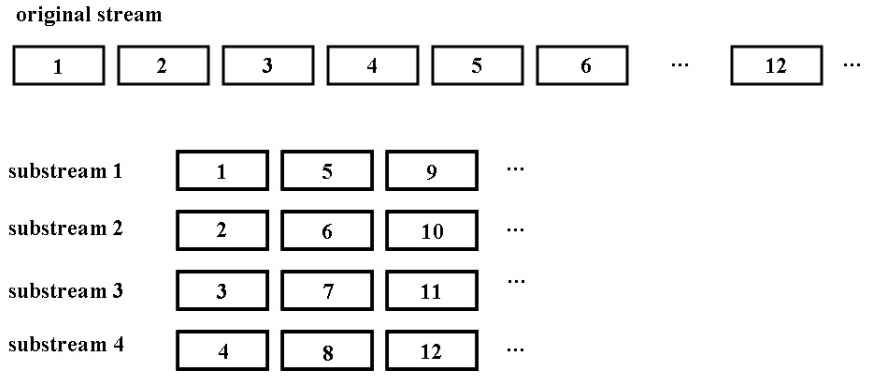

Fig. 6 Decomposition of the original video stream in several substreams (4 in the considered example)

The structure of the peer buffer has to be accordingly modified: a synchronization buffer precedes the cache buffer and hosts the chunks received from each substream, sequentially ordered, as the example in Fig.7 shows. The chunks are then combined

in a single stream as soon as chunks with contiguous sequence numbers are received from each substream.

In the example the video chunk with sequence number 11 is yet to be received and therefore the combination process halts at this point in the sequence.

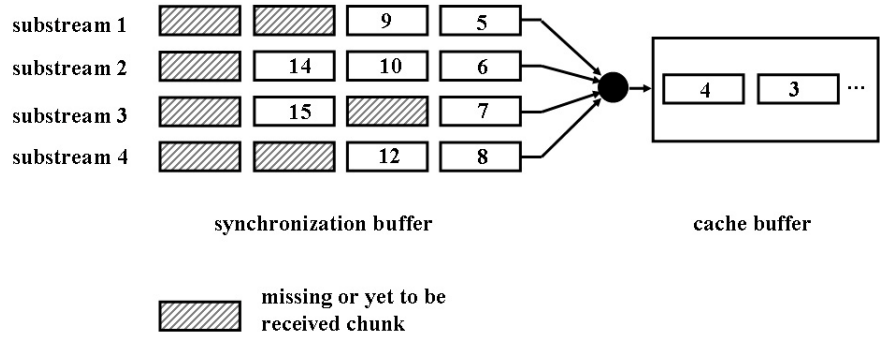

Fig. 7 The logical organization of the peer buffer when 4 substreams are considered

As for the new buffer maps exchanged by peers in the New CoolStreaming system, these too have undergone a significant rethinking: a buffer map now specifies not only the chunk availability at the peer, but also its current requests. In greater detail, the map is composed of two vectors, whose dimension is equal to K, the number of substreams: the elements of the first vector indicate the sequence number of the last chunks the peer received for every substream; the elements of the second vector specify what substreams the peer wants to subscribe to.

We close this shot with the description of the new content delivery mechanism adopted by CoolStreaming. As previously anticipated, peers in the original CoolStreaming prototype had to deliberately fetch – pull – each chunk from other peers. The revisited architecture inherits the pull mechanism only for the first chunk of the requested substream; from then onward, the parent peer will keep continuously forwarding – pushing – all chunks to the requesting node.

Periodically, parents peers are updated, to replace nodes that either departed, experienced a failure, or provided insufficient video content. We refer the interested reader to [7], as well as to [10], for details about the parent reselection process.

4 Popular P2P streaming systems

We have already picked up the popular P2P CoolStreaming architecture as the reference example; there are however several additional systems that are worth being cited, most notably because they experience wide commercial success and rely upon a large basis of users. Among them, we mention PPLive [11], SopCast [12], UUSee [13], GridMedia [14], offering real-time services, and Joost [15] and Ba-

belgum [16], offering Video-on-Demand (VoD) services. All of these systems are proprietary, have received a great success all over the world and their majority has been developed in China.

In what follows we briefly glide over their main features, beginning with PPLive [11]. It is one of the most popular P2P live streaming systems, born in China in 2004. It belongs to the mesh-based family, employs a pull-based protocol for video content transmission and mostly relies on TCP. It offers more than 200 channels, has an average of 400,000 daily users and the bit rate of its video programs ranges from 250 kbit/s to 400 kbit/s. It also offers a few channels at a rate of 800 kbit/s. Channels are encoded with two video formats: Window Media Video (WMV) or Real Video (RMVB) [17]. In PPLive the number of partner nodes of every peer depends on the popularity of the selected channel and on the peer's access type: peers with high bandwidth access (also termed campus peers) have about 40 partners; peers with residential access have a number of partners that ranges from about 10 to 30.

SopCast [12] is another P2P streaming application that provides both VoD and live services, born in China in December 2004. It was able to support more than 100,000 concurrent users only a few months after its introduction. It employs a mesh-based architecture and mostly relies on UDP. In contrast to PPLive, both residential and high bandwidth access peers typically download from 2 to 5 other peers.

UUSee [13] is an additional instance of very large scale P2P streaming solution and was born in China too. It has several streaming servers around the world, simultaneously offers more than 800 channels, with 100,000 concurrent users, and provides a streaming rate of 400 kbit/s. It belongs to the mesh-based family and employs the pull-based approach. It relies on TCP for data transmission and the number of partners for each user is around 50.

GridMedia [14] is still a Chinese large scale P2P live streaming system (there is no doubt that China get's the lion's share in this field!), implementing a hybrid push-pull protocol. It is able to support more than 224,000 simultaneous users and its streaming rate is 300 kbit/s. The streaming server can support up to 800 connections [18], therefore reaching 240 Mbit/s outgoing bandwidth at server side.

Finally, there are two interesting P2P Video-on-Demand (VoD) streaming solutions, both conceived in Europe: Joost and Babelgum.

Joost [15] is a VoD P2P system for distributing TV content. It was created by Niklas Zennstroem and Janus Friis, the Skype founders, in 2006. It relies on a mesh-based P2P streaming overlay and every peer receives 95% of the video frames from about 25 peers. Joost employs mostly UDP as transport protocol. In particular, UDP is used to transmit data packets and TCP for control messages only. An important feature of Joost is the adoption of a NAT detection mechanism in order to improve system performance: if some peers lie behind the same NAT device, they tend to transmit video content to each other.

Babelgum [16] is a Video-on-Demand P2P streaming system too, conceived by Silvio Scaglia in 2005. It employs TCP for both control and data packets distribution. Every peer receives 95% of the video frames from about 8 peers.

Both systems, Joost and Babelgum, exhibit a hybrid architecture. Their behavior is not purely P2P: rather, they resort to the help of some streaming servers located around the world to distribute the video contents.

5 Measurements and quality monitoring

In this Section we are going to present a few significant measurements about some large P2P commercial systems, namely, PPLive, SopCast and CoolStreaming. We will also examine a small P2P architecture, featuring a moderate number of users.

Both measurements performed at network edge and at a central facility will be introduced and commented: the former are attained client-side, where the single peer itself can capture the local upload and download traffic and analyze them; the latter are performed directly server-side. The log-server has a complete view of the system: to cite a few significant records, number of peers connected, session durations, upload/download bandwidth of each peer, peers' IP address/port number.

There are several reasons for measuring and monitoring system parameters: perhaps the most important is to guarantee a satisfying viewing experience to system users. This is an essential feature for wide commercial success of any P2P streaming systems, that has to guarantee video playback continuity, without video freezing and skipping, in order not to discourage the users. If a user is unsatisfied he/she could decide to abandon the system. Therefore, the main goal is to optimize as much as possible the quality perceived at the client side, the so-called Quality of Experience (QoE).

Different factors could threaten the QoE: participating nodes heterogeneity, frequent peers churn, delays. In what follows we explain the most significant IP-TV quality metrics, that in turn have a deep impact on QoE: initial start-up delay, video switching delay, video playback continuity.

The *start-up delay* is the time interval between the instant a channel is selected and the instant when the video playback starts on the screen. This is a critical delay, with a straightforward influence on the viewing experience of the users.

If the user decides to watch another video, he/she switches channel and this causes the *video switching delay*, which is longer than that experienced in traditional television.

As for *video playback continuity*, each video chunk has a playback deadline. Hence, if a certain video chunk is not in the buffer before its playback deadline, two situations can occur. If there are no video chunks in the buffer of the player, the user experiences freezing of the video, that is the playback of last video frame. If there are some video chunks in the buffer, the player plays them back although they might be not continuous: in this circumstance the user experiences skipping of the missed chunk's frames [4].

We now provide a glimpse at some specific attributes of the examined P2P systems, as can be inferred from local measurements. Next, their behavior and partly

their performance will be discussed, with the help of the measures available at the central facility.

5.1 Network edge measurements

This subsection focuses on measurements performed client-side, displaying some of the features exhibited by PPLive, SopCast, as well as by an instance of a small P2P overlay.

The first measurements illustrate what transport protocol the different P2P streaming systems adopt. Not all streaming architectures employ the same transport protocol: some of them exclusively utilize TCP, others UDP only, and others both. In the latter case, the system typically uses TCP for control traffic and UDP to carry data packets.

In order to know what kind of transport protocol a system relies upon, it is sufficient to check how many TCP and UDP packets are exchanged among the local peer and its partner nodes. The results we obtained for SopCast and PPLive are graphically shown in Figs. 8 and 9 respectively. We can conclude that SopCast employs UDP to transport both video and control packets. In contrast, PPLive employs UDP for data and TCP to carry control traffic. Unlike UDP, TCP guarantees reliable packet delivery and enforces congestion control: this first feature is useful for signaling traffic, but not for live video traffic, that has strict time restrictions.

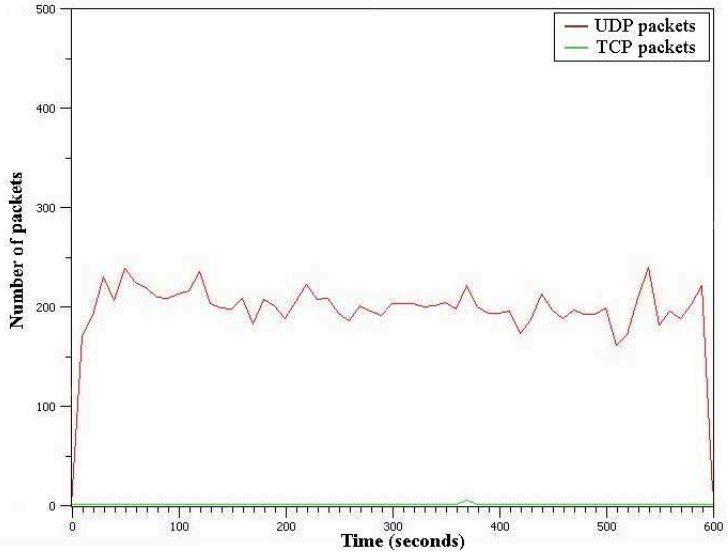

Fig. 8 Number of UDP and TCP packets exchanged in a time window of 600 s by a SopCast peer

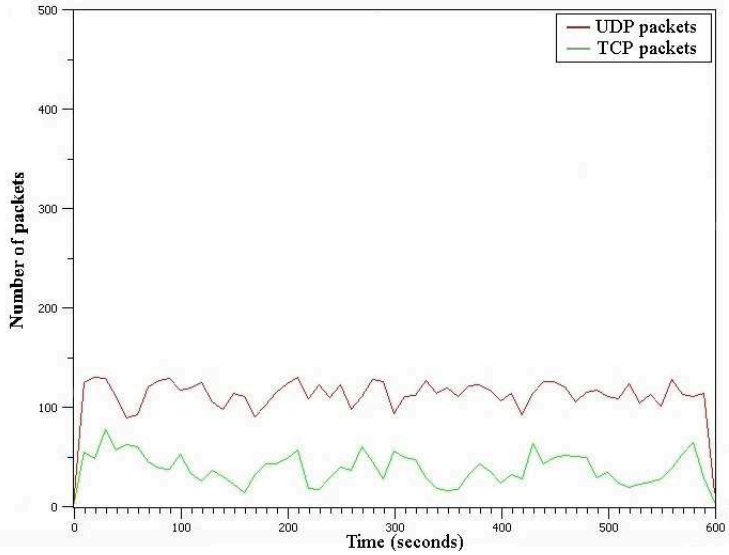

Fig. 9 Number of UDP and TCP packets exchanged in a time window of 600 s by a PPLive peer

We now turn our attention to the P2P application throughput: to locally assess it, we once more resort to local traffic captures, performed client-side. The values of the upload and download throughput that PPLive achieves are graphically represented in Fig.10 for an ADSL connection. As expected, the ADSL upstream and downstream asymmetry directly reflects in the throughput values the P2P application exhibits. To be accurate, the traffic exchanged by the local peer and captured to execute this analysis contains not only video chunks but also control messages, useful to manage peer's partnerships, and, in particular, buffer maps. Yet, control packets represent a far smaller percentage with respect to data.

There are no great differences between PPLive and SopCast, whose channels have both a streaming rate of about 400 kbit/s, so we choose not to report the results referring to the latter system.

Next measurements are concerned with the download throughput that the local peer achieves via its best partners, i.e., the peers from which it receives more packets. The results obtained for PPLive are shown in Fig.11: this figure reports the total throughput of the local peer, as well as the contributions provided by the best four partners and by the best partner. We can observe that the local peer receives about one third of the total chunks from the best four partners and much less from the best partner. This suggests that in PPLive every peer has to connect to several other peers, among which the chunk requests are equally allocated.

Until now, we have presented some results referring to large systems, featuring a vast group of users. However, there exist P2P live video streaming solutions belonging to the small overlay category, that are equally worth being investigated.

We focus our attention on one of them, exhibiting a small number of peers and providing only a few TV channels. Its client software is in the beta version and the floor of users is therefore quite moderate. It still provides few channels and their rate is around 160 kbit/s, with a resulting relatively low video quality. Privacy agreements restrain us from diffusing the P2P provider name and the system architecture details. However, we can say that it is a pure mesh-based P2P solution.

Here too, we take a look at a few interesting sets of local measures, namely:

- the number of partner nodes from which the peer receives data packets;
- the Cumulative Distribution Function (CDF) of the received packets' length.

The first measurement is useful to understand how many peers cooperate with the user to visualize the requested video. Fig.12 shows the result of this analysis. We can see that in one hour of experiment there are on average 5 other peers, that provide the local peer the desired data packets. At first, the peer receives data only from two peers, one of which is the streaming server. Then, as it starts receiving buffer maps, the local peer contacts more partners to request – and obtain – the chunks it needs.

The second experiment aims at providing an indicative characterization of traffic the peer receives from and sends to the P2P network. Fig.13 graphically shows the Cumulative Distribution Function (CDF) of the packet size, as derived from the local data. We can observe that the CDF follows a bimodal distribution, underlining two kinds of packets: large and small packets. The first ones convey data, have a size larger than 1500 bytes and sum up to 40% of the total number of packets. The second ones carry control information (conveying, e.g., buffer maps) and have a length that ranges from approximately 60 to 200 bytes. Although derived for the

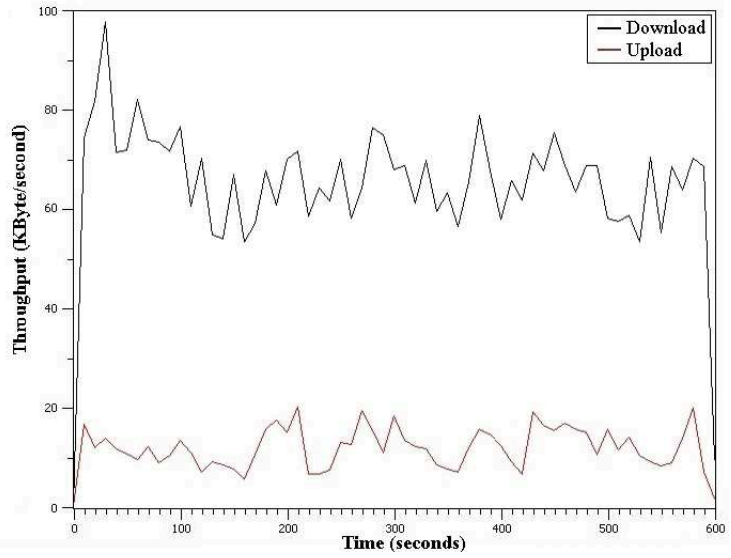

Fig. 10 Download and upload throughput of a PPLive peer

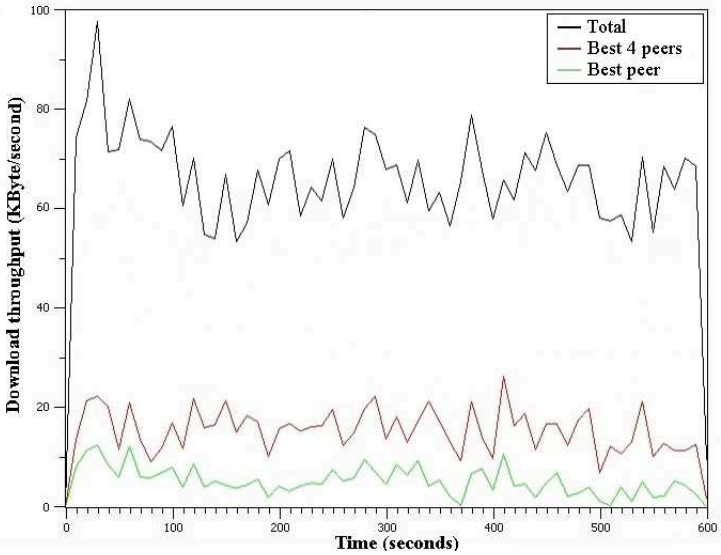

Fig. 11 Total and partial download throughputs of a PPLive peer

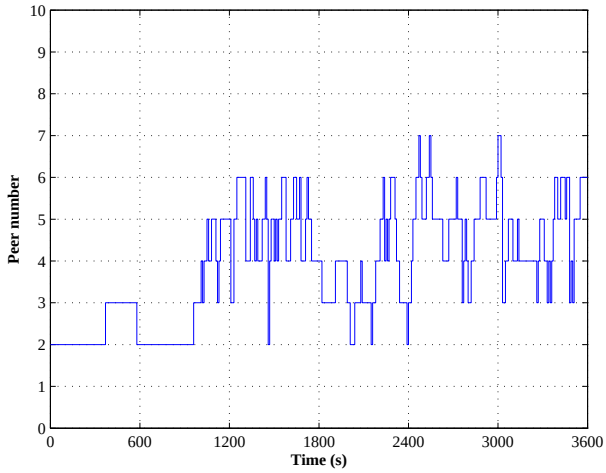

Fig. 12 Evolution of the number of peer's partners in a small system

small P2P overlay, it is important to note that the bimodal distribution is typically observed in large systems too.

5.2 System measurements

In this subsection we will comment some of the measurements that are typically performed server-side. The examined systems will be CoolStreaming, PPLive and the small P2P overlay considered earlier. The main reference sources will be [10] for CoolStreaming and [17] for PPLive, as log-server traces for these systems are not publicly available.

One of the most common measurements carried out on P2P video streaming systems is the number of peers joining/leaving the system and the number of peers contributing in data streaming transmission evolution during time. Fig.14 qualitatively reports a typical behavior for the number of simultaneous CoolStreaming users, as reported in [10], during the peak hours between 06:00 and 11:00 PM. These results underline the great scalability of the P2P solution, which easily supports more than 40,000 concurrent users.

This type of measurement has been performed on PPLive too [17]. The number of peers for one of its popular television channels is qualitatively represented in Fig.15. By observing the figure, we can say that the examined program reaches a large number of concurrent users, about 2,700. Moreover, the peaks of the users occur at 12 AM and 7 PM, suggesting that people tend to watch IP-TV *outside* office hours [17].

As expected, the number of concurrent users in the system is tightly correlated to the popularity of the program. For this reason, the authors of [17] have performed the same measurements during the Spring Festival Gala on Chinese New Year too, that is the most popular event in China. The results obtained emphasize a sudden increase of the users from 50,000 to 200,000 when the corresponding program starts,

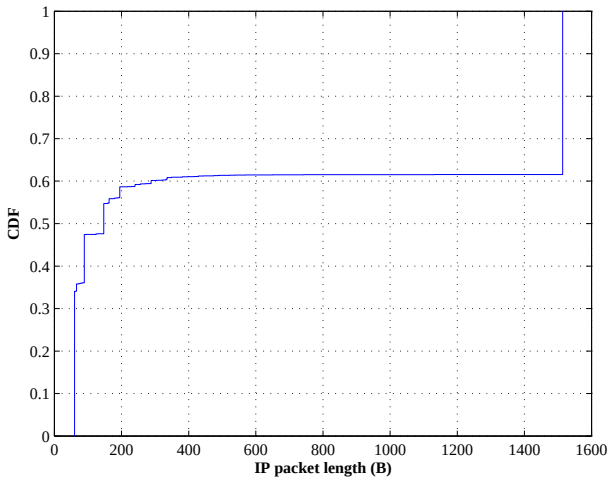

Fig. 13 Packet size CDF in a small system

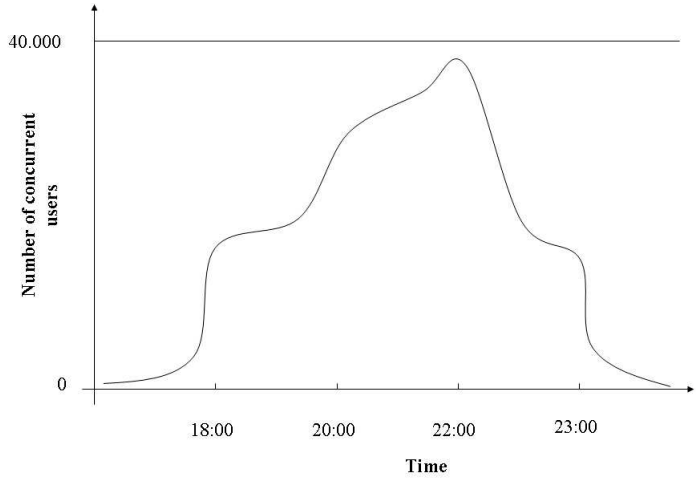

Fig. 14 Evolution of the number of simultaneous users in CoolStreaming from 18:00 to 23:00

and a sudden decrease when the program ends. Again, these observations suggest that the P2P system scales well. An important feature, common to all P2P streaming systems, is the following: when a TV program ends, peers immediately and simultaneously leave the network, so that a batch-departure occurs. This phenomenon is not present in P2P systems for file sharing, where users depart at different instants [17].

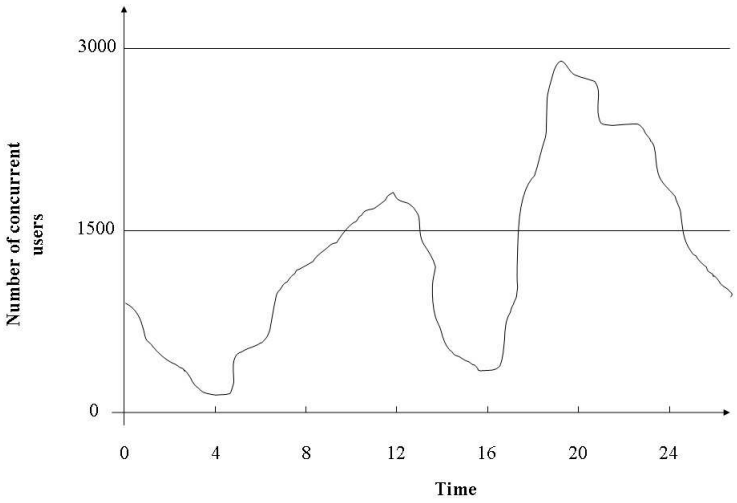

Fig. 15 Evolution of the number of simultaneous users in PPLive in a whole day

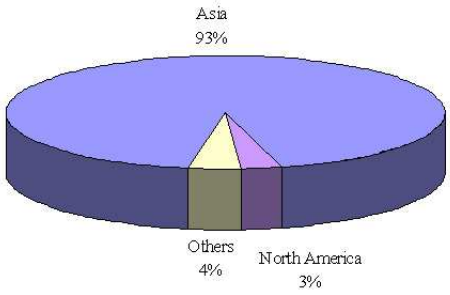

Fig. 16 Users geographic distribution in PPLive in a generic weekday

A large scale P2P streaming system typically attracts millions of users from all over the world. Therefore, another relevant feature to monitor is the geographic distribution of the users. This measurement requires a comparison between every peer's IP address and a database containing all the associations between ranges of IP addresses and corresponding geographic areas. The results reported for PPLive in [17] are shown in Figs. 16 and 17, that show the user geographic distribution in a generic weekday and in the Spring Festival Gala day on Chinese New Year, respectively. In Fig.16 we can see that the highest percentage of users are from Asia. In Fig.17 the situation is a bit different: the percentage of users from outside Asia is higher during this event, indicating that PPLive is able to attract hundreds of thousands of users from all over the world when important events are broadcasted.

Classifying user connections on the basis of their upload capacity is a further, useful distinction to perform. We can distinguish users into private, when their IP addresses are not visible outside their own LAN, and public, when their IP addresses are visible. If some peers have the same IP address, they are typically users behind

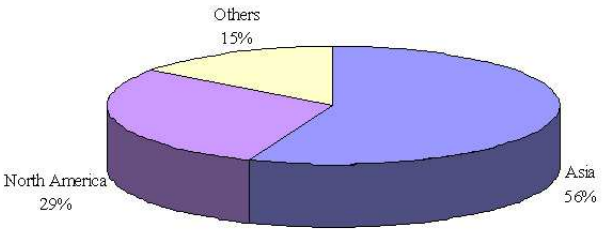

Fig. 17 Users geographic distribution in PPLive during the Spring Festival Gala

a NAT device. By checking the user capacity to establish TCP connections a peer can be further categorized as [10]:

- *Direct-connect*: peer with public address, that can establish partnerships to and from other peers;
- *Universal Plug and Play (UPnP)*: peer with private address, that can establish partnerships with other peers and the other peers can establish partnerships with it;
- *NAT*: peer with private address, that the other peers cannot establish partnership with;
- *Firewall*: peer with public address, that the other peers cannot establish partnership with.

The CoolStreaming analysis performed in [19] provides the peers classification shown in Fig.18. We notice that only a small percentage of the peers are UPnP and direct-connect nodes: they are 30% of the total and contribute with more than 80% to the upload bandwidth [19]. On the other hand, there is a significant percentage of users behind NAT devices, having limited uploading capacities. The remaining small percentage represents peers that usually stay in the system for a short period of time. Most of them are NAT/firewall peers [19]. We anticipate here that a similar situation occurs in the small overlay we have examined.

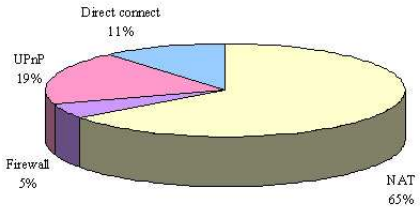

Fig. 18 Peer classification in Coolstreaming

An additional system feature that is worth mentioning is the session duration time, i.e., the time that elapses between the join and the departure of the peer from the system.

The results reported in [10] for the CoolStreaming system are shown in Fig.19, where the qualitative distribution of the session duration is shown. We observe that, once the users successfully obtain the video stream, they are stable and remain within the system for the entire program duration [10]. But there are also many short sessions, that depend on the startup failures of newly joined nodes.

Last parameter we comment upon is the start-up delay. Fig.20 plots its Cumulative Distribution Function (CDF) as reported in [10] for the native, pull-based only CoolStreaming scheme, as well as for the new CoolStreaming, based on the

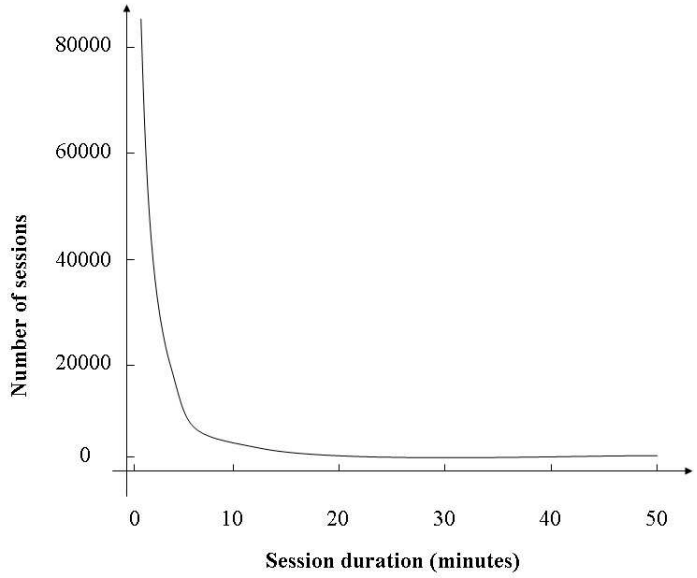

Fig. 19 Distribution of sessions duration in Coolstreaming

hybrid push-pull architecture. It is immediate to conclude that, although the pull-based system is simple and robust, the hybrid solution definitely exhibits a superior performance.

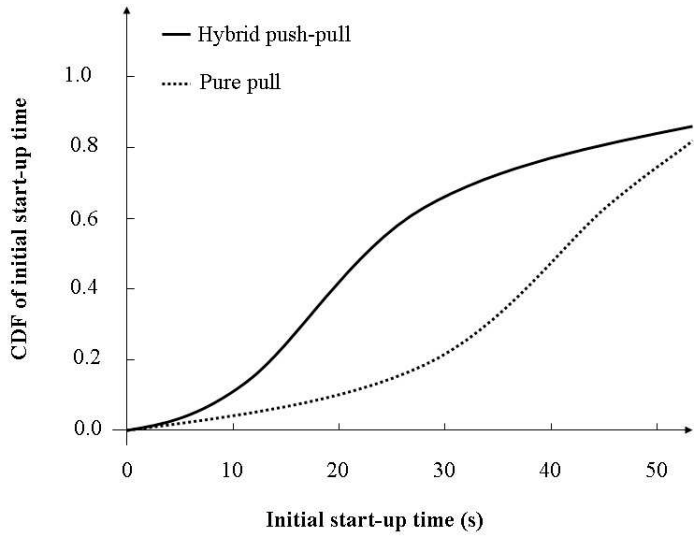

Fig. 20 Start-up delay CDF in Coolstreaming

In order to shed some light on how a small system works, we next present and discuss some measurements performed server-side on this kind of architecture. The parameters that have been monitored are:

- the number of users in the weekday and during peak hours;
- the geographic distribution of the users;
- the percentage of free riders upon the total number of peers.

Fig.21 plots the number of users in system during a soccer match of the 2008 European Championship. The number of concurrent users remains fairly low until about 14:00, it then increases reaching the peak (around 1300 peers) when the match begins. This number progressively decreases at the end of the match. It is immediate to notice that the number of users of this small system is by far lower than in CoolStreaming and PPLive.

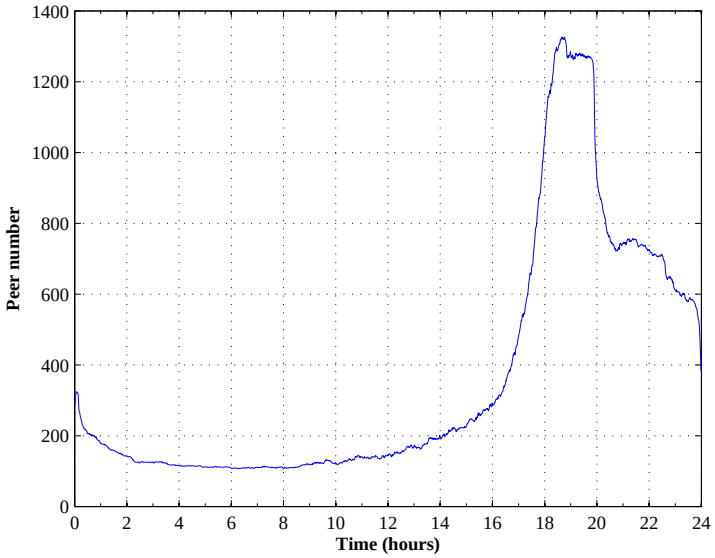

Fig. 21 Evolution of the number of peers during a very popular event in a small system

Next, Fig.22 represents the geographic distribution of the users in this small overlay. We can observe that most of them are located in Europe, but there is also a non negligible amount of U.S.A. nodes. The remaining percentage represents users from the rest of the world.

A last, important feature that we take into consideration is the presence of free riders, without any distinction between nodes lying behind a NAT device or a firewall. These peers do not contribute to the diffusion of the video stream, as they receive chunks from others, without uploading anything. Free riders therefore represent a serious threat to the functionality of any P2P system. As Fig.23 shows, their

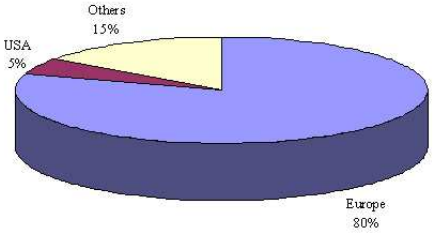

Fig. 22 Geographic distribution of peers in a small P2P system

number is significant in the small overlay, where their negative impact on performance has to be properly limited. If no countermeasures are taken, the risk is to unbearably degrade the QoE level that "good", collaborative users expect.

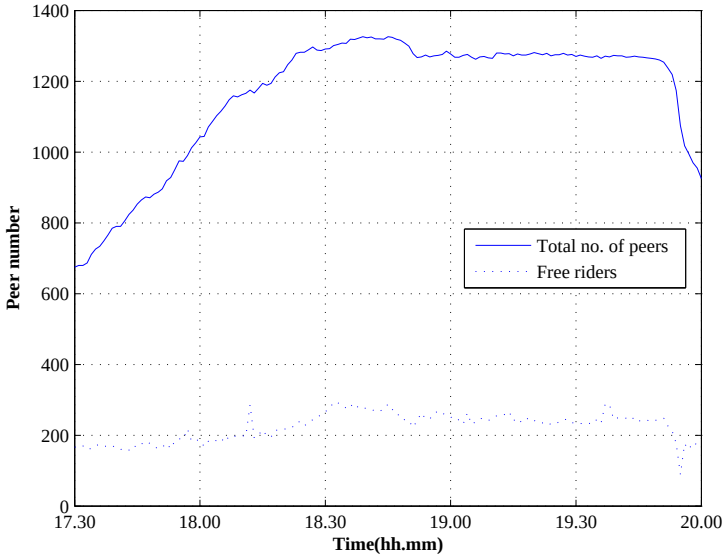

Fig. 23 Total number of users and free riders in a small system

6 Modeling insights

Now that we have understood how P2P streaming systems work and how their behavior can be monitored, we shift our attention to some of the most significant contributions in the field of modeling that recently appeared in literature. The goal is to delineate the effects on performance of the main system attributes: upload and download capacity of the peers; classes of peers and their cardinality; buffering available at peers and bearable play-back delay; free riders and churns that dynamically and unpredictably spoil system equilibrium.

6.1 First modeling efforts

To begin with, we develop some simple considerations that help understand the advantages of the P2P solution with respect to the traditional, centralized client-server architecture.

Following [20] and more substantially [21], we investigate how the server-side capacity u_s needed to support a streaming rate r to N identical users can be decreased, when each peer makes available a fraction η of its upload bandwidth b_u to the system.

In this simple, deterministic setting, we observe that

$$u_s = max[0, N(r - \eta \cdot u)]. \tag{1}$$

The relation between N and u_s is depicted in Fig.24, for $r = 700$ kbit/s, an upload bandwidth $u = 400$ kbit/s and for different values of η, namely, $\eta = 0.9, 0.5$ and 0. From visual inspection, it is straightforward to conclude that when the P2P system is well designed, a considerable saving is achieved in server capacity: as an example, when $N = 2 \times 10^4$ peers populate the system, u_s can be decreased from 14 Gbit/s to 6.8 Gbit/s for $\eta = 0.9$.

Next, inevitable question is: how can the efficiency η be computed?

To answer, we resort to [21]: the focus of this work is on P2P for file sharing, rather than for video streaming, yet its analysis can be easily adapted to our case as well. In details, this work assumes that the upload bandwidth of a peer will not be utilized only if all of its partners already have the chunks of that peer: it follows that the efficiency η is

$$\eta = P[\text{there is at least one peer that wants a chunk the peer has}]. \tag{2}$$

If K is the number of partner peers and the distribution of chunks between peers is independent from peer to peer, and identical for all peers, then

$$\eta = 1 - P[\text{peer j needs no chunks from peer i}]^K \tag{3}$$

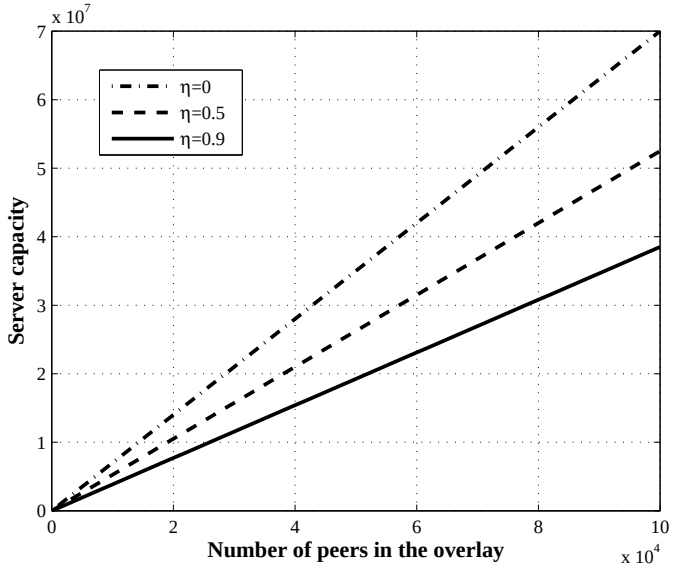

Fig. 24 Server capacity to support N simultaneous viewers in the P2P architecture ($\eta \neq 0$) and in the client server approach ($\eta = 0$)

We next denote by n_i the number of chunks that peer i possesses, out of the M available for sharing in the peer buffer, and assume n_i is uniformly distributed in $[0,\ldots,M-1]$. We can therefore write:

$$\begin{aligned} &P[\text{peer j needs no chunks from peer i}] = P[\text{peer j has all pieces of peer i}] = \\ &= \textstyle\sum_{n_j=1}^{M-1} \frac{1}{M} \sum_{n_i=0}^{n_j} \frac{1}{M} \cdot P[\text{peer j has all chunks of peer i}|n_i,n_j] = \\ &= \frac{1}{M^2} \sum_{n_j=1}^{M-1} \sum_{n_i=0}^{n_j} \frac{\binom{M-n_i}{n_j-n_i}}{\binom{M}{n_j}} = \frac{1}{M^2} \sum_{n_j=1}^{M-1} \frac{\binom{M+1}{n_j}}{\binom{M}{n_j}} = \\ &= \frac{1}{M^2} \sum_{n_j=1}^{M-1} \frac{M+1}{M+1-n_j} = \frac{M+1}{M^2} \sum_{n_j=1}^{M-1} \frac{1}{M+1-n_j} = \frac{M+1}{M^2} \sum_{m=2}^{M} \frac{1}{m} \end{aligned} \tag{4}$$

having set $m = M+1-m_j$.

Replacing last outcome in η expression, it can be concluded that

$$\eta = 1 - \left(\frac{M+1}{M^2}\right)^K \cdot \left(\sum_{m=2}^{M} \frac{1}{m}\right)^K . \tag{5}$$

The (K, η) relation, illustrated in Fig.25 for three different values of the buffer size M, reveals that it is useless to increase the number of partner peers K above 5-10, as efficiency has already reached unity. It also suggests that the number of available

chunks M in the peers' cache plays a significant role when determining η: when M is large, η increases. On the other hand, when this number grows, the play-back delay that the peer experiences also increases, a phenomenon that needs to be accurately monitored. Interestingly $5-10$ is deemed an adequate partner range for several of the current P2P IP-TV applications.

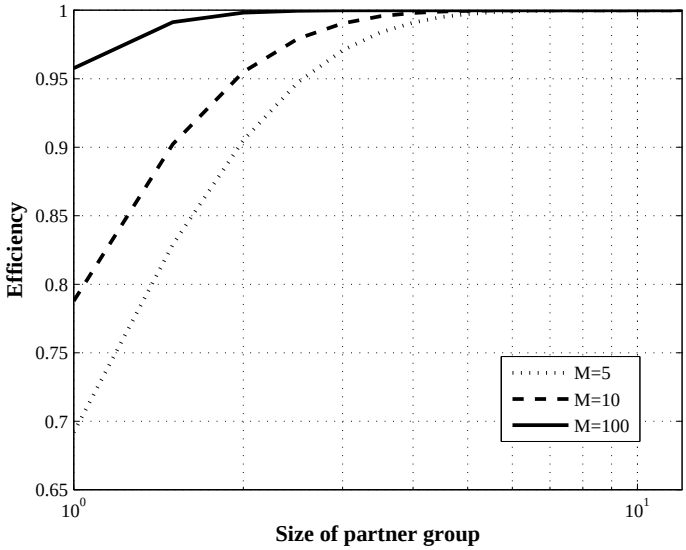

Fig. 25 Efficiency as a function of the parent group size K, for different values of the number M of available chunks in the peers' buffer

6.2 More recent contributions

A deeper insight into the fundamental characteristics of P2P streaming systems that rely upon a fully connected mesh is achieved in [22].

To understand its contribution, let us introduce some proper assumptions and notations.

A fluid model is employed to describe the video stream that propagates from the server at rate r, and to describe the bits – as opposed to chunks – that propagate among peers. The system initially examined is bufferless, i.e., bits are not cached in the peer before being played back or before being copied to other peers.

Let N be the number of peers in the system and let u_i denote the upload bandwidth of the i-th peer, $i = 1, 2, \ldots, N$. When all peers in the system receive the video at rate r, the system is said to guarantee *universal streaming*.

The first result that is provided states that the maximum achievable streaming rate, r_{max}, is given by

$$r_{max} = min\left\{u_s, \frac{u_s + \sum_{i=1}^{N} u_i}{n}\right\} \tag{6}$$

that implies *all* peers contribute to the swarming process with their *entire* upload bandwidth (equivalently, the efficiency η_i is 1, $\forall\, i$, $i = 1,2,\ldots,N$).

For most real P2P systems, a useful reference is a two class model, where a user can be classified as either a super peer or an ordinary peer: typically, the former has a high-speed access, such as from a campus network, whereas the second utilizes a residential, ADSL access: we assume all peers in either class exhibit the same upload capacity. We denote by N_1 and N_2 the number of super peers and ordinary peers, respectively, and by u_1 and u_2 their upload capacities. A further hypothesis is that $u_2 < r < u_1$.

It immediately follows from (6) specialized to the latter case, that universal streaming can be achieved whenever the following inequality is satisfied:

$$r \leq \phi(N_1,N_2) = min\left\{u_s, \frac{u_s + N_1u_1 + N_2u_2}{N_1 + N_2}\right\} \tag{7}$$

Let us now examine in greater detail the $\phi(N_1,N_2)$ term.

Let peers of both classes join the system following a Poisson process, of rate λ_i, $i = 1,2$, and indicate by μ_i, $i = 1,2$, the rate at which they leave. No hypothesis is made about their sojourn times, that can be arbitrary. Let $N_1(t)$ and $N_2(t)$ indicate the number of peers of the two classes in the system at time t: note that they are two independent Poisson random variables [23].

Focusing on the second term in $\phi(N_1,N_2)$, which is the significant one for all the cases of practical interest, we can compute the probability of achieving universal streaming as

$$P\left[r \leq \frac{u_s + N_1(t)u_1 + N_2(t)u_2}{N_1(t) + N_2(t)}\right] = P\left[N_1(t) \geq cN_2(t) - u_s^{'}\right], \tag{8}$$

where

$$c = \frac{r - u_2}{u_1 - r} \quad \text{and} \quad u_s^{'} = \frac{u_s}{u_1 - r}. \tag{9}$$

Recalling that $N_1(t)$ and $N_2(t)$ are independent Poisson random variables, hence obey the following distribution,

$$f_i(n) = \frac{e^{-\rho_i}\rho_i^n}{n!} \quad with \quad \rho_i = \frac{\lambda_i}{\mu_i} \quad and \quad i = 1,2, \tag{10}$$

a few passages lead to the conclusion that

$$P[\text{universal streaming}] = F_2(M) + \sum_{l=M+1}^{\infty} \left((1 - F_1(\left\lceil cl - u_s' \right\rceil) + f_1(\left\lceil cl - u_s' \right\rceil) \right) f_2(l) \tag{11}$$

where

$$F_i(n) = \sum_{l=0}^{n} f_i(l) \quad \text{and} \quad M = \left\lfloor \frac{u_s'}{c} \right\rfloor . \tag{12}$$

Making use of (11), it is now possible to explore the achievable performance of the P2P system. The work in [22] examines "small" overlay, with a number of simultaneous peers in the proximity of 75, and a "large" overlay, whose number of simultaneous peers are in the proximity of 7500. The considered rates are $r = 3$ units, $u_1 = 7$ units and $u_2 = 1$ units, to be interpreted, e.g., as 300, 700 and 100 kbit/s, respectively. The average sojourn times are set to $\frac{1}{\mu_1} = \frac{1}{\mu_2} = 30$ minutes for both classes; in the small overlay λ_2 is set to 100 peers per hour, so that ρ_2, the average number of ordinary peers, turns out to be equal to 50; in the large overlay, λ_2 is set to 10000 peers per hour, leading to $\rho_2 = 5000$. In both scenarios λ_1 is then varied: in the proximity of 25 in the small system, near 2500 in the large one.

Having defined the probability of degraded service as

$$P_{degr} = 1 - P[\text{universal streaming}] , \tag{13}$$

Fig.26 qualitatively reports P_{degr} as a function of the ratio $\frac{\rho_1}{\rho_2}$, determined in [22] for both a small and a large overlay, when $u_s = 7$: this figure shows that in both systems performance improves when the arrival rate of super peers increases (equivalently, when the $\frac{\rho_1}{\rho_2}$ ratio increases). It can further be observed that the P2P system with a large population performs better, as the large overlay provides universal streaming over a much wider range of system parameters: the physical justification behind this behavior is that the departure of super peers has a confined effect in the large overlay, whereas it can lead to a significant worsening in the small system.

The study also points out that when the system scales towards larger dimensions, the role of the server and its upload rate u_s become more and more marginal.

Once the assumptions of bufferless peers is removed, the analytical, closed-form approach has to be abandoned in favor of numerical solutions obtained via a simulation tool. The most salient outcome derived by the authors of [22] is that the buffer introduction brings a remarkable improvement to system performance. For instance, they observe that the large overlay previously examined, that achieved a probability of degraded service equal to 0.5 at $\frac{\rho_1}{\rho_2} = 0.5$, experiences a P_{degr} of approximately 0.05 when buffering is introduced. Moreover, a cache that can store 30 seconds of the video is already sufficient to exploit almost all the potential buffering gain. Analogous results hold for the small overlay.

Finally, it is observed that buffering can even bring a larger improvement than increasing the infrastructure server bandwidth u_s.

What is the limit of the analysis just described? Mainly the assumption that all N peers are connected to each other and that all N-1 peers contribute with their available content to the stream the remaining peer is viewing: definitely not what

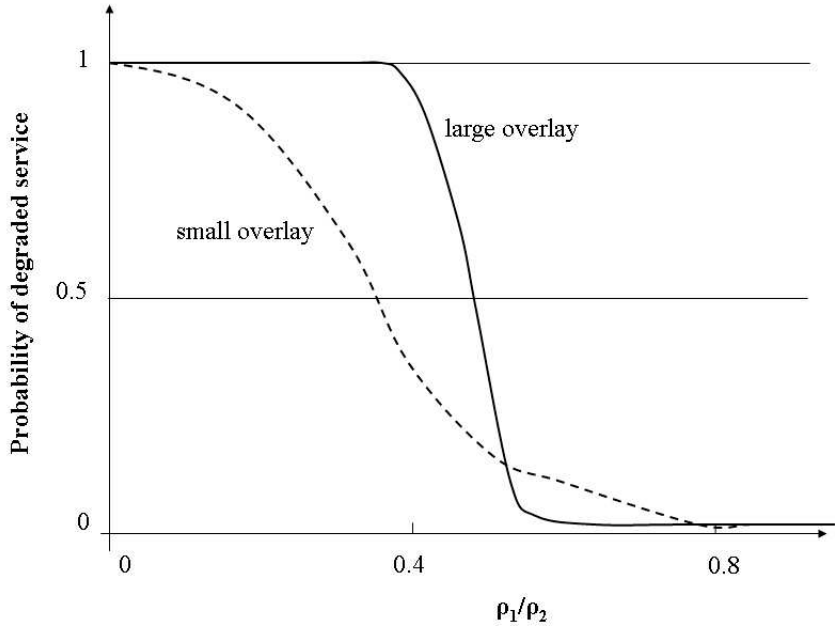

Fig. 26 Probability of failing to achieve universal streaming in a small and in a large overlay, as a function of $\frac{\rho_1}{\rho_2}$, when $\lambda_2 = 100$

happens in a real system, as we previously described. Being able of utilizing all the peer upload bandwidth it is not only a matter of random fluctuations in the peer numbers, $n_1(t)$ and $n_2(t)$!

To mention only a few additional key elements, we observe that a successful system behavior heavily depends on the scheduling algorithm employed by peers, on the parents they rely upon, on the type of connections that peers experiment (they can be fully visible or belong to the free rider class, meaning that they can download content but do not contribute to the system with any upload bandwidth).

Yet, the main goal of [22] is to determine the *maximum* achievable streaming rate r_{max} and correspondingly tie the evaluation of the probability of degraded service to it, and as such, it represents a genuine effort.

There are a few, additional papers that deal with the performance evaluation of P2P streaming systems. Although we do not have room to cite here all their outcomes, yet we believe that they deserve a careful citation: [24] for the theoretical effort in finding heuristics with provable good performance; [18] for the mathematical analysis that helps comprehend why the pull-based approach is so efficient in utilizing peer upload capacity; [25] for a model relying on stochastic graph theory, able to capture the fundamental properties of the mesh-based systems. The interested reader is strongly encouraged to enrich his/her knowledge through these excellent references.

6.3 A numerical comparison between mesh and multiple-trees

If we now abandon the analytical world and consider the simulative approach, an interesting comparison between mesh-based and multiple tree-shaped overlays is performed in [26].

In this work, the hypothesis that both systems share is the adoption of Multiple Description Coding (MDC). For this technique, a video stream is divided into multiple substreams: each can be independently decoded and the quality of the received video increases with the number of decoded substreams. Every peer can subscribe to a different number of substreams, depending on its download bandwidth. The MDC choice helps in counteracting the heterogeneity in peers' bandwidth, and therefore warrants a better bandwidth usage.

In the multiple tree approach, different substreams propagate via disjoint trees; each peer can join different trees, subject to the constraint that it has to be an internal node in only one tree, and has therefore to appear as a leaf in the remaining trees of the overlay. Moreover, the trees have to be balanced and short, meaning that the number of their internal nodes as well as their depth have to be comparable.

Proper rules are set to handle node arrivals and departures [26]: as an example, a new peer is added as an internal node to the tree with the lowest number of internal nodes; a new internal node is placed as a child for the node with the lowest depth that can accept a new child or has a leaf child; when an internal node departs, the subtree that was rooted in it tries to rejoin the tree as a whole, but if it does not succeed, its constituent nodes independently rejoin the tree.

In the examined mesh-based approach, the content delivery mechanism is very similar to the one BitTorrent adopts. A new peer contacts the bootstrapping node, that replies providing a random subset of peers able to act as parents for the node; connected peers have a parent-child relationship, like in the tree-based architecture. Peers periodically indicate what video chunks they have available to their child peers and in turn request – pull – chunks from their parents.

The packet scheduling aims at profitably utilizing the upload bandwidth that peers make available to the mesh-based system; it also strives to guarantee an adequate quality of the received video allowing for the reception of several substreams; it finally guarantees a timely delivery of the requested video chunks.

The crucial difference between the two approaches, correctly outlined by the authors of this paper, lies in the delivery process of the substreams. Whereas in the multiple-tree overlay all chunks belonging to one substream *statically* follow the same tree to propagate among peers, in the mesh overlay the delivery tree is individually chosen for each chunk and is *dynamically* shaped as the chunk crosses the network. This greatly helps at efficiently employing all the available bandwidth of the peers. In contrast, in the multiple tree overlay there might be occurrences where one of the internal peers does not have sufficient upload bandwidth "to feed" all its child peers.

In terms of similarity between the two architectures, the first perspective that [26] takes is to state that the superposition of multiple trees is essentially equivalent to a mesh. Another similarity is that in both architectures video chunks follow a tree

to reach their destination: it is manifest in the multiple-tree overlay, but even in the mesh-based case it is possible to capture the snapshots of the per-chunk trees, i.e., of the delivery paths that each single video chunk follows [5]: these are indeed trees, whose internal nodes are quite stable and whose depth is typically longer compared to tree-based systems, mainly because of their dynamic formation. Additionally, the per-chunk trees exhibit a high degree of correlation, sharing several internal links.

The content delivery mechanism of both push and pull-based architectures is numerically examined in [26] by simulation: initially, 200 homogeneous peers with symmetric upload and download bandwidths are considered, in a system that employs 20 substreams, each coded at a rate $bw_d = 80$ kbit/s. The *access link* bandwidth of each peer is set equal to $K \times deg \times bw_d$, where K, $K \geq 0$, is a tunable parameter and *deg* indicates the degree of each peer, i.e., the number of incoming and outgoing connections the peer can support. When *deg* is the same for all peers, $K \times bw_d$ is the average *per-connection* bandwidth. The physical topology underlying the examined systems, as well as the adopted congestion control protocol, is detailed in [26].

The study determines the percentage of bandwidth utilization over the entire system and the average quality delivered to each peer: the first metric quantifies the effects of the content bottleneck phenomenon, i.e., of the situation where a parent peer does not possess any useful chunk to deliver to a particular child, even though it has some upload bandwidth available, as well as the effects of a low link access bandwidth at the peers; the second performance index is defined as the average number of substreams a peer receives during a session and gives an idea of the quality of the received video.

The system bandwidth utilization is determined as a function of the average per connection bandwidth, $K \times bw_d$: in the mesh-based system its behavior is practically independent of the peers' bandwidth, and it takes on high values, of the order of 0.95, revealing that the mesh always allows to fully leverage the upload bandwidth that the peers make available to the system. In the multiple tree-based architecture, when the average per-connection bandwidth is low, system utilization is poor: this is caused by parent peers that cannot satisfyingly support all their downstream connections in the trees; when the per-connection bandwidth is high, system utilization is low too, although for a different reason: the content bottleneck phenomenon appears. In between these two extremes, satisfying values of bandwidth utilization are achieved, that are however slightly lower – of the order of 0.9 – with respect to the alternative system: they are obtained in the proximity of $K = 1$, i.e., when the per-connection bandwidth is slightly higher than the the description bandwidth bw_d. Fig.27 illustrates in a qualitative manner this behavior.

Next, the behavior of the average received quality is investigated. As Fig.28 indicates, the quality always increases with the per-connection bandwidth in the mesh system, in a practically linear manner; in the multiple tree approach, increasing the per-connection bandwidth causes the average quality to reach the target value of *deg*, but this limit is not trespassed.

It can be concluded that the mesh-system can fruitfully exploit any value of peer bandwidth, delivering a proportionally higher quality.

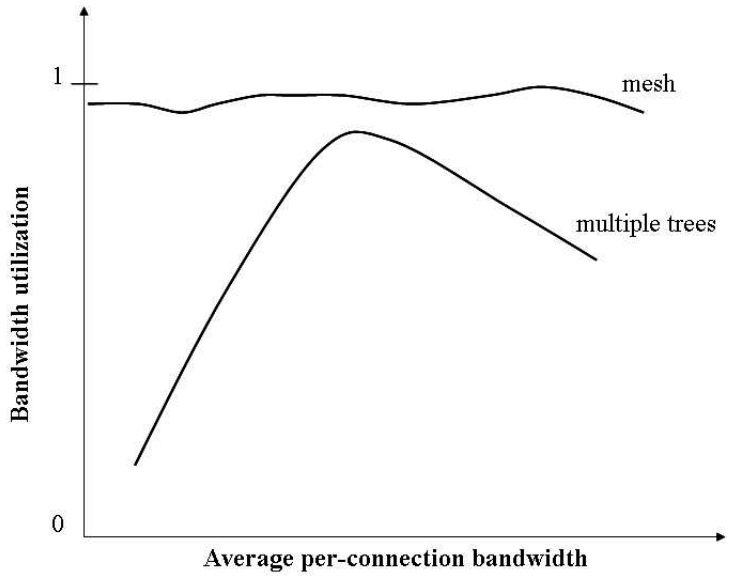

Fig. 27 The qualitative dependence of system bandwidth utilization on the average per-connection bandwidth

When the number of peers that a chunk visits before reaching the destination peer is computed, the simulations reveal that the average path length is longer in the mesh-based approach, as expected.

When the effect of bandwidth heterogeneity among peers is taken into account, both architectures display better utilization and quality.

In contrast, when the effect of the overlay size is examined, the study indicates that the mesh successfully scales, whereas the bandwidth utilization and the quality of the multiple tree approach gradually worsen. This can be explained by the fact that with no changes in the degree, the depth of the trees increases.

As for the ability to cope with churns, this comparative study assumes that the duration of the peer session is lognormally distributed, and that the peer interarrival times obey a Pareto distribution, with properly set parameters [26].

It is observed that the path from source to individual peers is more stable in the mesh, and that for both approaches the ancestor changing rate increases with peer population, due to the fact that the average distance between peers increases with peer population.

In conclusion, this numerical investigation provides useful insights when the main focus is on bandwidth utilization and quality; with respect to these performance metrics, its outcomes indicate that the mesh-based architecture consistently achieves superior performance. However, the study only partially investigates the delays introduced by the mesh architecture, a delicate issue, as also [7] and [18] have evidenced.

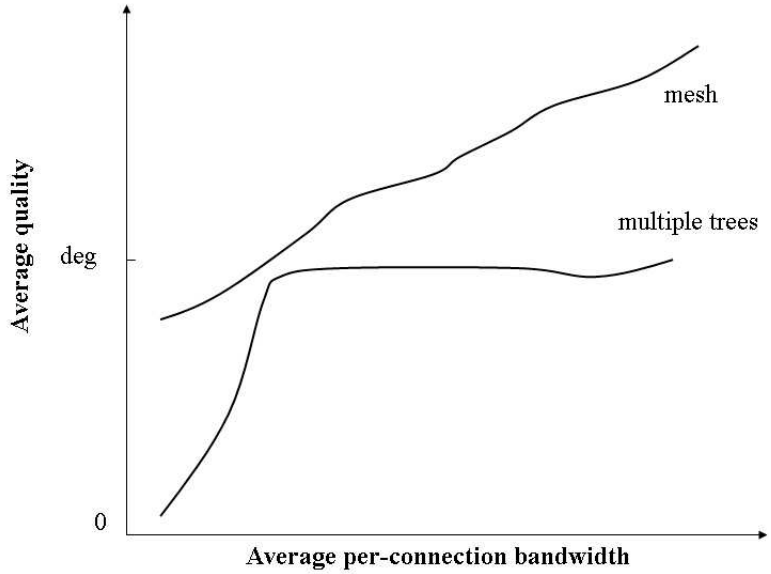

Fig. 28 The average received quality as a function of the per-connection bandwidth

7 Open issues and promising solutions

A wide, large-scale deployment of P2P architectures to distribute live video streaming over the Internet still requires sound answers to several challenging questions: we provide a brief "to-do list" in what follows.

On the QoS/QoE side, the scarce control that P2P systems have on the quality the viewers endure is undoubtedly an issue, mainly in highly dynamic environments. This limit is intrinsic to the structure of the current Internet; nevertheless, it needs to be adequately tackled in the light of commercial, pervasive adoption of P2P-based solutions.

Directly related to the QoE issue, there lies the necessity to decrease the start-up delays, that are often in the order of several tens of seconds: definitely a new, undesired experience for viewers of ordinary TV channels, that on the contrary often and rapidly change channel. As [17] indicates, possible solutions to investigate are network coding and redundant downloading. Another suggested approach requires the employment of dedicated servers, that provide the video at relatively low quality, whereas the P2P overlay guarantees the high quality video [4]. When a peer joins the system, it first – and quickly – receives the low quality stream from the server, then begins to effectively employ the P2P network to obtain the video at a higher quality.

Also, smaller playback lags are needed, to avoid the weird situation where the frames that some peers view are minutes behind other peers. This calls for efficient chunk scheduling techniques and more intelligent peering strategies.

On the network side, a crucial point is represented by the impact that P2P traffic has on ISP infrastructures: simple peer selection and random scheduling usually place a significant burden on the underlying network, and peers are highly spread, as the geolocalization analysis shows. Network-aware resource allocation within the overlay should substantially help in relieving the strain.

Finally, users lying behind NAT and firewalls may not be able to contribute to the system with their upload bandwidth: better NAT traversal schemes have to be put into use; how to treat such users, what QoE they should receive is also an interesting dilemma. The work in [3] suggests some simple schemes to improve the overlay construction when peers lie behind a NAT: we report them in Fig.29. As shown in Fig.29 (a), one of the peers could take on the responsibility of broadcasting to all other users behind the NAT; alternatively, the incoming P2P traffic could be confined, yet some redundancy be guaranteed, as Fig.29(b) summarizes.

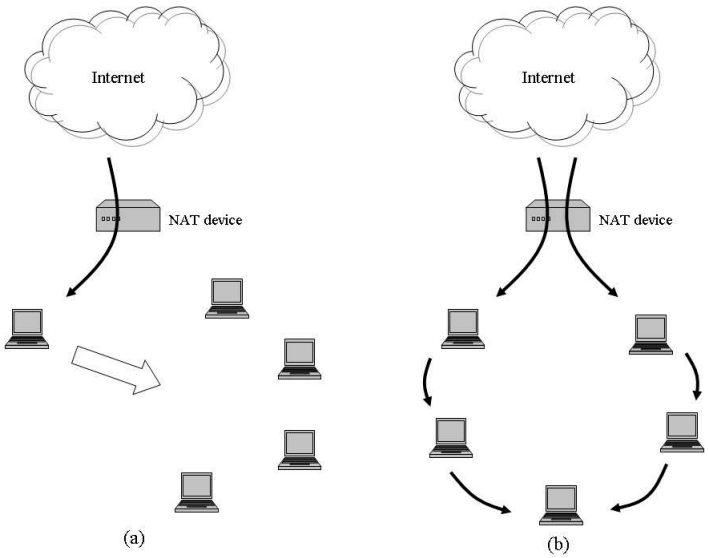

Fig. 29 (a) A possible optimization scheme for NAT users; (b) An alternative solution

On the measurement side, the remote monitoring of P2P systems is a further, significant chapter: [27] represents a good example of how to track playback continuity, start-up latency and playback lags in a network-wide manner relying upon the harvesting of buffer maps. However, this is "only" passive monitoring: proactive countermeasures are still to be devised, to assure and protect a satisfying viewing experience.

We dot not have room to explore in detail all these interesting questions; rather, we have intentionally decided to concentrate on a specific, promising instance of enhancement, that is, network coding. For doing so, we refer to [28] and [29], notwith-

standing that there are several alternative approaches that are equally worth being investigated.

The first step to understand the potential of network coding for P2P video streaming was performed in [28], where a conventional pull-based P2P overlay was directly compared with the analogous one enhanced by the adoption of random network coding. In the latter system, each video chunk $\mathbf{b}$ is further divided into n blocks $\mathbf{b} = [b_1, b_2, \ldots, b_n]$, each block b_i having a fixed size k. When a video chunk has to be transmitted to a peer, the parent, say p, rather than sending the entire chunk, chooses m blocks out of the n within the chunk, $m \leq n$, and a set of random coefficients $c_1, c_2, \ldots, c_m$ in the Galois field $GF(2^8)$, to produce one coded block x of size k:

$$x = \sum_{i=1}^{m} c_i^p \cdot b_i^p \ . \tag{14}$$

As each coded block is a linear combination of m original blocks, it is uniquely identified by the coefficients of the combination. Then, the parent peer p transmits the coded block, together with the coefficients of the linear combination (a overhead not to be underestimated).

As the session proceeds, a peer accumulates coded blocks from its parents and in turn encodes blocks to forward to its child peers. The destination peer d can recover the original video chunk as soon as it has received n linear independent coded blocks $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, taking advantage of the following relation:

$$\mathbf{b} = \mathbf{A}^{-1} \cdot \mathbf{x}^T \ , \tag{15}$$

where $\mathbf{A}$ is the matrix of the coding coefficients of x (each row in $\mathbf{A}$ corresponds to the coefficients of one coded block).

The decoding process becomes faster resorting to Gauss-Jordan elimination: now the peer starts to decode a video chunk as soon as it receives its first coded block. Moreover, if the peer receives a coded block that is linearly dependent on previously received blocks, the elimination process provides an all zeros row; in turn, this event triggers the discarding of the coded block, while the receiver keeps waiting for additional data. In other words, there is no need of an *a priori* check on the received blocks, to guarantee they are linear independent.

What the authors of the study in [28] observe is that the overlay with random network coding is more resilient against network dynamics and achieves a better bandwidth usage. This has to be ascribed to the finer granularity introduced by the coding mechanism in the streaming process: the system unit is the coded block, rather than the entire video chunk.

Also note that in this overlay a peer missing a video chunk may be served by multiple randomly selected peers that have coded blocks of the same requested chunk.

Last remark becomes crucial for the design of the novel P2P streaming system proposed in [29], and termed R^2.

Here too, random network coding is confined within the single chunk, for the primary reason of reducing the number of blocks that the encoder has to manipulate, therefore limiting its complexity.

However, to enhance the probability of having different multiple sources for the same video chunk, each contributing with its own coded block, this new system employs a random push, rather than the explicit pull requests that are typical of mesh-based systems, to drive the diffusion process.

In greater detail, it is the destination peer, say *d*, that – very frequently – advertises what its missing chunks are (rather than what its availability is), via its buffer maps. Now any peer *p* that receives this information and possesses any of these chunks can potentially act as one of the origins for that chunk. Indeed, the algorithm states that, if *p* possesses the chunk, *p* randomly decides whether to push out one coded block of the chunk that *d* is missing.

A video chunk is therefore truly served by multiple originating peers, and this choice allows to take full advantage of the benefits random network coding brings in. While the session proceeds, the receiving peer accumulates coded blocks from different contributing peers into its local buffer; as in [28], it immediately starts the progressive decoding process resorting to Gauss-Jordan elimination.

The reader is referred to [29] for the thorough description of R^2 and of all its characteristics and constituent algorithms.

The idea behind the R^2 proposal is depicted in Fig.30, that helps to understand how it departs from a traditional pull-based overlay, whose concept is shown in Fig.31.

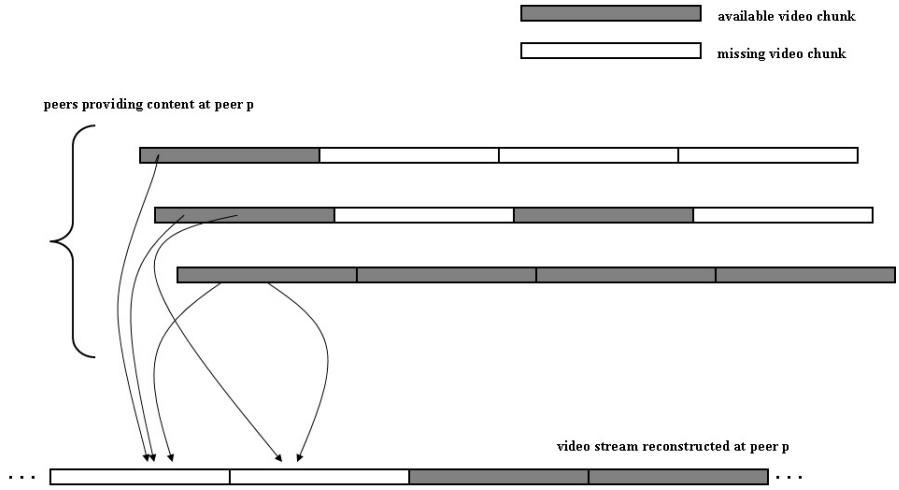

Fig. 30 The concept behind R^2 and its adoption of network coding

As intuition also suggests, the newly proposed architecture allows the adoption of much larger video chunks than pull-based overlays. This can be qualitatively explained observing that in a conventional system a missing video chunk is served by one partner peer at a time. In contrast, a missing video chunk in R^2 is typically sent by multiple originating peers, and each of them selects via a proper algorithm

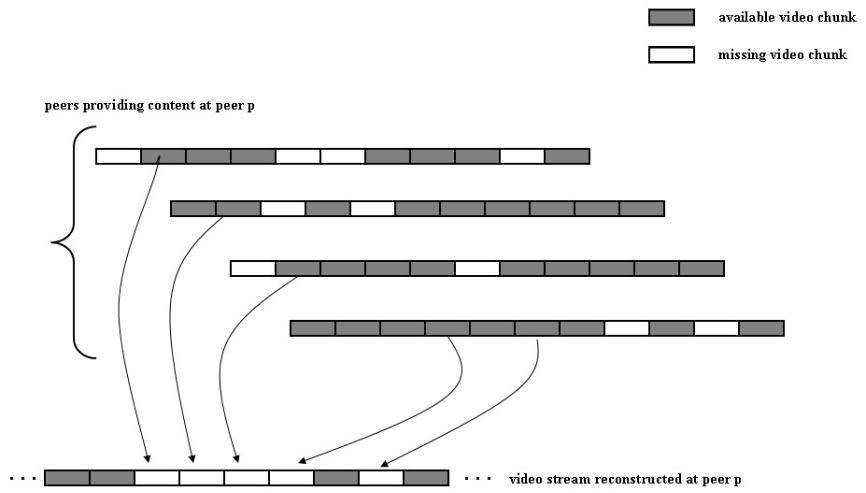

Fig. 31 The concept behind a traditional pull-based mesh system

(i) which chunk to send; (ii) what coding coefficients to use. Equivalently, peers collaborate, randomly and without any explicit knowledge of each other, to the reconstruction of each chunk.

Several additional assumptions are introduced in [29] and contribute to the satisfying performance of the system: to cite a few, playback is synchronized, so as to maximize the overlap of playback buffers in the peers; peers receive buffer maps in an extremely timely manner; when composing coded blocks, peers attribute higher priority to video chunks close to the – common – playback deadline. As a consequence, the study in [29] demonstrates that R^2, when properly tuned, outperforms conventional P2P solutions: it significantly reduces the number of playback skips in a streaming session, therefore achieving better playback quality; it quickly fills buffers at the peers at start up time and it maintains them adequately filled during the entire session, despite of peers' departures and arrivals. On the negative side, the new system has to be carefully tuned in terms of chunk and block size; it does imply complex choices, such as the synchronization of peers; it mandates buffer maps to be almost continuously exchanged and control messages to constitute a non negligible fraction of the entire traffic. Nevertheless, it truly represents the first step toward a deeper understanding of the potential of random network coding in P2P overlays.

References

1. Sentinelli, A., Marfia, G., Gerla, M., Kleinrock, L., Tewari, S.: Will IPTV ride the peer-to-peer stream? IEEE Communications Magazine, Vol.45, Issue 6, June 2007, pp. 86 - 92.
2. Liu, J., Rao, S.G., Li, B., Zhang, H.: Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. Proceedings of the IEEE, Vol.96, Issue 1, January 2008, pp. 11 - 24.

3. Zhang, X., Liu, J., Li, B., Yum, T.S.P.: Coolstreaming/DONet: a data-driven overlay network for efficient live media streaming. Proc. of IEEE Infocom 2005, 13-17 March 2005, Vol.3, pp. 2102 - 2111.
4. Hei, X., Liu, Y., Ross, K.W.; IPTV over P2P streaming networks: the mesh-pull approach, IEEE Communications Magazine, Vol. 46, Issue 2, February 2008, pp.86 - 92.
5. Wang, F., Liu, J., Xiong, Y.: Stable peers: existence, importance, and application in peer-to-peer live video streaming. Proc. of IEEE Infocom 2008, 13-18 April 2008, pp. 1364 - 1372.
6. Li, B., Yin, Y.: Peer-to-peer live video streaming on the internet: issues, existing approaches, and challenges. IEEE Communications Magazine, Vol. 45, Issue 6, June 2007, pp. 94 - 99.
7. Li, B., Xie, S., Qu, Y., Keung, G.Y.; Lin, C., Liu, J., Zhang, X.: Inside the New Coolstreaming: principles, measurements and performance implications. Proc. of IEEE Infocom 2008, 13-18 April 2008, pp.1031 - 1039.
8. Floyd, S., Handley, M., Padhye, J., Wiedmer, J.: Equation based congestion control for unicast applications. Proc. of ACM/SIGCOMM 2000, May 2000, Vol. 1, pp. 1 - 14.
9. Handley, M., Floyd, S., Padhye, J., Wiedmer, J.: TCP Friendly Rate Control (TFRC): protocol specification, RFC 3448, January 2003, ftp://ftp.rfc-editor.org/in-notes/rfc3448.txt, proposed standard.
10. Li, B., Xie, S., Keung, G.Y., Liu, J., Stoica, I., Zhang, H., Zhang, X.: An empirical study of the Coolstreaming+ system. IEEE Journal on Selected Areas in Communications, Vol. 25, Issue 9, December 2007, pp. 1627 - 1639.
11. PPLive, “PPLive Homepage,” http://www.pplive.com/
12. SopCast, “SopCast Homepage,” http://www.sopcast.com/
13. UUSee, “UUSee Homepage,” http://www.uusee.com/
14. GridMedia, “GridMedia Homepage,” http://www.gridmedia.com/
15. Joost, “Joost Homepage,” http://www.joost.com/
16. Babelgum, “Babelgum Homepage” http://www.babelgum.com/
17. Hei, X., Liang, C., Liang, J., Liu, Y., Ross, K.W.: A measurement study of a large-scale P2P IPTV system. IEEE Transactions on Multimedia, Vol. 9, Issue 8, December 2007, pp. 1672 - 1687.
18. Zhang, M., Zhang, Q., Sun, L., Yang, S.: Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better? IEEE Journal on Selected Areas in Communications, Vol. 25, Issue 9, December 2007, pp. 1678 - 1694.
19. Xie, S., Keung, G.Y., Li, B.: A measurement of a large-scale peer-to-peer live video streaming system. Packet Video 2007, November 2007, pp. 153 - 162.
20. Tewari, S., Kleinrock, L.: Analytical model for BitTorrent-based live video streaming. Proc. of IEEE CCNC 2007, Consumer Communications and Networking Conference, January 2007, pp. 976 - 980.
21. Qiu, D., Srikant, R.: Modeling and performance analysis of BitTorrent-like peer-to-peer networks. Proc. of ACM SigComm, pp. 367 - 378.
22. Kumar, R., Liu, Y., Ross, K.: Stochastic fluid theory for P2P streaming systems. Proc. of IEEE Infocom 2007, May 2007, pp. 919 - 927.
23. Kleinrock, L.: Queueing systems. Volume I: Theory. John Wiley & Sons, New York, 1975.
24. Massoulie, L., Twigg, A., Gkanttsidis, C., Rodriguez, P.: Randomized decentralized broadcasting algorithms, Proc. of IEEE Infocom 2007, May 2007, pp. 1073 - 1081.
25. Carra, D., Lo Cigno, R., Biersack, E. W.: Graph Based Analysis of Mesh Overlay Streaming Systems. IEEE Journal on Selected Areas in Communications, Vol. 25, Issue 9, December 2007, pp. 1667 - 1677.
26. Magharei, N., Rejaie, R., Guo, Y.: Mesh or multiple-tree: a comparative study of live p2p streaming approaches. Proc. of IEEE Infocom 2007, May 2007, pp. 1424 - 1432.
27. Hei, X., Liu, Y., Ross, K.W.: Inferring network-wide quality in P2P live streaming systems, IEEE Journal on Selected Areas in Communications, Vol. 25, Issue 9, December 2007, pp. 1640 - 1654.
28. Wang, M., Li, B.: Lava: a reality check of network coding in peer-to-peer live streaming, Proc. of IEEE Infocom 2007, May 2007, pp. 1082 - 1090.

29. Wang, M., Li, B.: R^2: random push with random network coding in live peer-to-peer streaming, IEEE Journal on Selected Areas in Communications, Vol. 25, Issue 9, December 2007, pp. 1655 - 1666.