TURINstream: A Totally pUsh, Robust and effIcieNt P2P video streaming architecture

TURINstream: A Totally pUsh, Robust and effIcieNt P2P video streaming architecture

Andrea Magnetto, Rossano Gaeta, Marco Grangetto, *Senior Member, IEEE,* and Matteo Sereno, *Member, IEEE*

***Abstract*—This paper presents TURINstream, a novel P2P video streaming architecture designed to jointly achieve low delay, robustness to peer churning, limited protocol overhead, and quality of service differentiation based on peers cooperation. Separate control and video overlays are maintained by peers organized in clusters that represent sets of collaborating peers. Clusters are created by means of a distributed algorithm and permit the exploitation of the participant nodes upload capacity. The video is conveyed with a push mechanism by exploiting the advantages of multiple description coding. TURINstream design has been optimized through an event driven overlay simulator able to scale up to tens of thousands of peers. A complete prototype of TURINstream has been developed, deployed and tested on PlanetLab. We tested our prototype under varying degree of peer churn, flash crowd arrivals, sudden massive departures, and limited upload bandwidth resources. TURINstream fulfills our initial design goals, showing low average connection, startup, and playback delays, high continuity index, low control overhead, and effective quality of service differentiation in all tested scenarios.**

***Index Terms*—Peer-to-peer, Video streaming, Multiple Description Coding, Push architecture, PlanetLab testbed**

I. Introduction

PEer-to-peer (P2P) streaming has proved to be a viable and efficient approach to support the broadcasting of live or pre-recorded media over the Internet. Peers contribute their uplink bandwidth by forwarding content to their connected peers. Global available resources (aggregate uplink bandwidth) grows as the number of peers increases thus making the approach potentially able to scale to a large number of users.

Nowadays P2P streaming architectures can be broadly classified in two classes: *tree-based* (e.g., [1], [2], [3]) and *mesh-based* (e.g., [4], [5]). In tree-based approaches (also termed as push-based) the overlay is composed of several diverse trees that are used to multicast the video packets to the peers. The tree based approach can easily exploit Multiple Description Coding (MDC) [6], [7], where the video is encoded onto a set of independent descriptions to be pushed across separate trees. The mesh-based approach is based on file swarming mechanisms where participating peers form a randomly connected mesh and use gossip-like protocols for the creation and the administration of the overlay; buffer maps are frequently exchanged among the peers to signal the available video chunks. Each peer aims at retrieving the video stream by explicitly requesting its missing chunk (pull-based approach). Recently [8] provided a comparison between the two approaches as well as the identification of similarities and differences. Push/tree P2P system generally guarantees lower startup and playback delays, which depend on the topological features of the overlay. Despite these advantages the results in [8] indicate that the mesh-based approach generally exhibits a superior performance over the tree-based approach. The main shortcomings of the tree-based systems are reported to be the large overhead due to the organization of the tree topology, the difficulty to respond to the dynamics of the peers, and the suboptimal exploitation of the upload bandwidth. Nevertheless, the mesh-based approach suffers from other problems; Although several mesh-based mechanisms that deployed MDC have been proposed [9], [10], the use of MDC techniques is not straightforward since one has to guarantee that the descriptions follow independent distribution trees; indeed the average decoded video quality is maximized if one assumes independent losses of the descriptions. Access control policies and service differentiation are difficult to implement. Furthermore, mesh-based approaches suffer from the tradeoff between control overhead and delay. In fact, to minimize delay peers must notify their neighbors of available packets as frequently as possible, thus resulting in high control overhead. On the other hand, to reduce control messages notifications must be aggregated over time thus making the delay higher. An attempt to bring the benefits of tree-based approaches in the mesh-based scenario is presented in [11]. The authors propose a pull-push hybrid protocol where packets are pushed along the trees, dynamically created according to a pull-based protocol. In [12] another mesh based system that uses a push data dissemination approach is presented.

In the present paper, we take a different viewpoint. In particular, we leverage on the topological guarantees of a tree structured overlay yielding short delays and allowing to exploit MDC video, while improving on the side of robustness to peer dynamics and bandwidth utilization. These two latter objectives have been addressed with the introduction of a small set of fully connected collaborating peers forming what we termed a *cluster*. The contributions of this work are the design, simulation and implementation of TURINstream, a novel multi tree P2P streaming architecture where tree nodes are represented by *clusters* of peers. The concept of cluster has already been introduced in some previous research; as opposed to such related works, briefly recalled in Sect. V, where the clusters are organized in a hierarchical structure, the overlay topology proposed in this paper is very flexible and permits a more efficient exploitation of the resources.

The authors are with Università degli Studi di Torino , Dipartimento di Informatica, Torino, Italia. E-mail: {magnetto,gaeta,grangetto,sereno}@di.unito.it

Separation of data and control overlays, cluster based organization, exploitation of MDC techniques, and tree-based control overlay are all exploited in the TURINstream design to achieve:

- high degree of robustness providing playback continuity in order to minimize freezing or blackout periods despite peers departures, failures, and churning. When a peer leaves or crashes, the cluster does not disconnect from the network and, while the cluster is being repaired, the video content keeps reaching the cluster members. At the same time, the probability of a joint failure of all the peers supplying the video to a cluster becomes negligible as the number of multiple trees increases;
- low connection, startup and playback delays. These metrics all depend on the depth of the control tree whose scaling is logarithmic with respect to the total number of participating peers;
- high scalability and low control overhead to allow a very large number of users to join the application;
- rewards for altruistic (cooperative) users and penalties for selfish (un-cooperating) ones. In fact, the join algorithm we define allows for the adoption of access control policies in terms of bandwidth and peer fairness; in other terms, differentiated quality of services policies can be used to pay back more collaborative peers, i.e., peers that share a higher uplink bandwidth.

The TURINstream design has been aided by an event driven overlay simulator able to scale up to tens of thousands of users. Finally, the major contribution of the paper is the development of a complete prototype of TURINstream which has allowed us to deploy and test the application on PlanetLab. The TURINstream architecture has been used to stream H.264/AVC video encoded with the MDC technique presented in [7]. Such MDC codec is compliant with H.264/AVC and has allowed us to use the standard RTP [13] protocol to transport the video packets and RTSP/SDP [14], [15] protocols to signal the video session. We tested our prototype under varying degree of peer churn, flash crowd arrivals, sudden massive departures, and limited upload bandwidth resources. The main finding is that TURINstream fulfills our initial design goals, showing low average connection, startup, and playback delays, high continuity index, low control overhead, and effective quality of service differentiation in all the scenarios we addressed.

The outline of the paper is as follows: Sect. II presents the protocol operations, Sect. III describes the simulator we developed while Sect. IV presents the TURINstream prototype and the tests performed on PlanetLab. Finally, Sect. V discusses similar approaches in the literature while Sect. VI concludes the paper with the outline of some of the future developments.

II. Protocol Description

In this section we describe the TURINstream protocol. The protocol is fully decentralized and it is designed to define and maintain separated control and video streaming topologies that are graphically shown in Fig. 1; this feature allows to keep the video stream flowing across the peers while the control overlay is undergoing some rearrangements in reaction to peer exits/crashes and bandwidth fluctuations.

A. Terminology and notation

A *peer* P is a host participating to the overlay.

The node that injects the video stream in the network is called *root source*; the proposed protocol can use multiple root sources improving in terms of robustness and scalability, e.g. by streaming each description from a different source. The node that coordinates the root sources and that is responsible for maintaining their neighborhood relationships is called *tracker*. The tracker and the root sources can reside on the same host.

The key element of the proposed protocol architecture is the *cluster*. A cluster C is a group of up to N cooperating peers (set of nodes enclosed by a dash circle in Fig. 1), coordinated by a peer *outside* C called *cluster-head*.

In our protocol, each peer belongs to only one cluster $\mathit{cluster}(P) = C$. We use $\mathit{members}(C)$ to denote the set of peers belonging to C. Similarly, we denote as $\mathit{headed}(C)$ the set of clusters whose cluster-head is a member of C; every cluster is fed with video data by one or more external peers called *sources*. The cluster-head is constrained to be one these sources. Every peer of C receiving one video packet from a source must share such packet with the remaining members of its cluster.

As for the video codec, we assume to use MDC; therefore the stream (whose bitrate is R) is composed by D ($D > 1$) independent and complementary *descriptions*. Each description can be decoded independently yielding the base video quality; such quality is improved by decoding more descriptions and depends only on the number of the received descriptions. Each description is further subdivided in L sub-streams called *stripes* each requiring a bitrate $r = \frac{R}{L \cdot D}$. A stripe is identified by the pair (d, l) where $0 \leq d < D$ and $0 \leq l < L$. While MDC works at the video coding level and generates independent and mutually refinable video streams, the stripe is introduced only at transmission level by selecting the packets of one description based on their counter identifier modulo L. The stripe represents the minimum access unit to the video stream. This approach is followed to allocate the upload bandwidth contributed by each peer with a finer granularity with respect to the number of descriptions.

Every peer can share a portion of its upload bandwidth $U(P)$ for both intra-cluster and inter-cluster communications. To simplify the management of the upload bandwidth of the peer we introduce the concept of *slot*, defined as the bandwidth required to upload a single stripe. The total amount of slots available at peer P, i.e. the maximum number of stripes that P can upload, is given by $S(P) = \lfloor \frac{U(P)}{r} \rfloor$. At a given time instant, each peer will be characterized by a certain slot allocation $S(P) = S_I(P) + S_E(P) + S_F(P)$, where $S_I(P)$ and $S_E(P)$ are the slots used for intra and inter cluster communications respectively, and $S_F(P)$ represents the free slots available at peer P.

Please note that most of previous definitions are time dependent; we explicit the time dependency only where it is

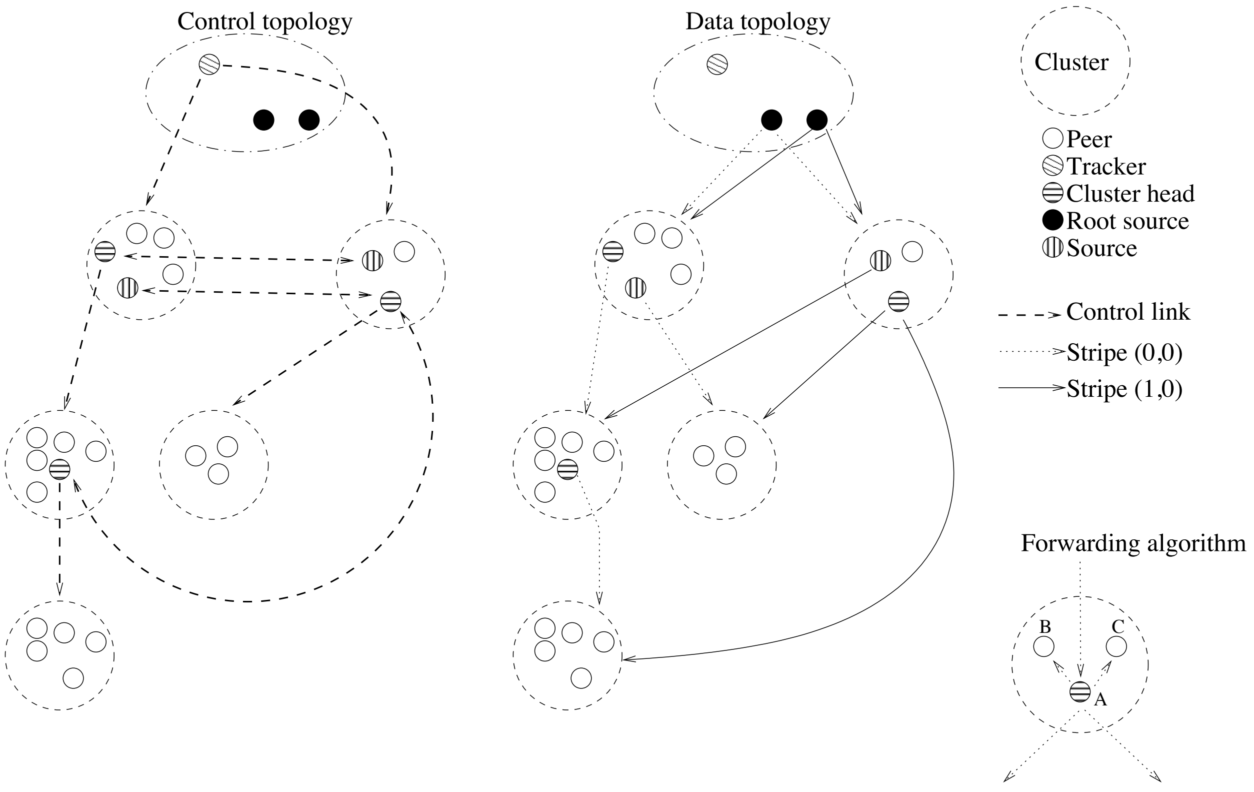


Fig. 1. TURINstream control and data topologies.

TABLE I
TABLE OF SYMBOLS

Symbol	Description
D	number of video descriptions
L	number of stripes per description
R	video stream bitrate
r	bitrate for each stripe
P	a generic peer
C	a generic cluster
N	maximum number of peers per cluster
$U(P)$	upload capability of peer P
$S(P)$	slots of peer P
$S_I(P)$	slots for intra-cluster communications
$S_E(P)$	slots for inter-cluster communications
$S_F(P)$	free slots

strictly necessary. To ease the task of the reader the notation used in this paper is summarized in Tab. I.

B. Data delivery overlay

A peer can decode the video at full quality provided that its cluster receives all the $D \cdot L$ stripes. Nonetheless, the use of MDC guarantees service continuity, despite the lower level of fidelity, when a subset of the descriptions is received. TURINstream distributes the video stream to the clusters by building and maintaining a video data distribution overlay that is a forest of trees, one for each stripe. In Fig. 1 an example of multi-tree for the distribution of 2 descriptions and one stripe per description is depicted.

1) Data sources for a cluster: For a given cluster C each stripe is provided to *members*(C) by a source that does not belong to the cluster. Each packet is forwarded to a single member P of C, that in turn propagates the information to all its cluster-mates. For each packet of a given a stripe (d, l), the recipient P is selected within C in a round robin fashion among the peers that have enough upload resources according to the bandwidth management algorithm described in Sect. II-C5. It is now possible to define the set of cluster sources for C as $\mathit{sources}(C) = \{P | \exists_{d,l} \mathit{source}(C, d, l) = P\}$ as well as the set of clusters receiving stripe (d, l) from peer P as $\mathit{served\_clusters}_{(d,l)}(P) = \{C | \mathit{source}(C, d, l) = P\}$, where $\mathit{source}(C, d, l)$ is the peer pushing the stripe (d, l) to the cluster C.

To better exploit the advantages of MDC TURINstream aims at guaranteeing that a source can serve a cluster for stripes belonging to only one description, to make most likely that if a source stops serving a cluster, this impacts only on the delivery of one description until a new source will replace the departed one. Formally, $\forall_{S \in \mathit{sources}(C)} \exists_{d,l} \mathit{source}(C, d, l) = S \Rightarrow \forall_{d' \neq d, l'} \mathit{source}(C, d', l') \neq S$. This topological feature is enforced as far as possible thanks to the bandwith allocation algorithm described in Sect. II-C5. Indeed, it is of paramount importance in order to make the delivery paths of the video descriptions as independent as possible so as to maximize the average quality of the received video.

2) Cluster level: Every cluster is characterized by its distance from the source root, defined as *level*(C). The level of a cluster C does not change over time and represents an upper bound to the number clusters to be traversed by any path of the video data distribution graph from the root source to C. The root source has level 0, clusters in the set $served\_clusters_{(d,l)}(root)$ have level 1 and so on. On the contrary, a peer can move across the clusters and its level changes over time in response to network dynamics as described in the following.

We impose that the sources of a cluster C must belong to clusters whose level is lower than $level(C)$, i.e., $\forall_{S \in sources(C)} level(S) < level(C)$. This constraint guarantees that the video packets of each stripe are forwarded along acyclic paths. Nevertheless, we do not impose $level(S) = level(C) - 1$ obtaining a more flexible data overlay structure, which in turn makes more likely to find spare upload resources during overlay rearrangements. An example of data link between levels 1 and 3 is shown in Fig. 1. Since a level 1 cluster is constrained to receive the video stream directly from the root source, then it follows that the number of such clusters cannot exceed $\lfloor \frac{U(\text{root source})}{R} \rfloor$.

3) Forwarding algorithm: Content distribution occurs from sources to clusters. Upon reception of a data packet in stripe (d, l) a peer $P \in C$ is responsible for intra-cluster sharing. All members of C are guaranteed to receive all packets of each stripe if

$$\sum_{P' \in C} (S(P') - S_E(P')) \geq DL\,(|\mathit{members}(C)| - 1) \qquad (1)$$

holds; in fact, $S(P') - S_E(P')$ represents the maximum number of slots contributed by P' and $DL\,(|\mathit{members}(C)| - 1)$ slots are needed for intra cluster video distribution, given that DL slots are provided by the external $\mathit{sources}(C)$.

In case P is also source for the higher levels, then it must forward the packet using inter-cluster communications. The main problem is how to select one peer P' for each cluster in $\mathit{served\_clusters}_{(d,l)}(P)$. In particular, one source selects P' depending on the forwarding activity already performed by P' and on the slots allocation described in Sect. II-C5.

In Fig. 1 (bottom-right corner) an example of packet forwarding involving three peers is shown; node A is sent a packet from a lower level cluster, then A forwards it to its served clusters and its cluster members B and C. Let us assume that the next packet for the same stripe hits B; now, B will share it with C and A, that in turn will forward it downward in the tree. From previous example it is clear that each video packet takes two hops to traverse a cluster in the worst case. Since every cluster is fed by only one peer for a given stripe it follows that each stripe spans along a tree rooted at the root source; the complete overlay for video data delivery is a forest of trees, one for each stripe.

C. Control topology

A control infrastructure is required to organize and maintain peers in clusters as they join and leave the application. To this end, one of the sources of a cluster is also responsible for the management of the cluster and it is called *cluster-head*. The control topology is composed by the edges from cluster-heads to the peers belonging to the clusters they head. Every connected peer depends only on one cluster-head. We will refer to the cluster-head of P as its *father* and to the members of the cluster as the cluster-head's *children*. Since the role of the cluster-head can be played by any of the sources, the control topology is usually different from any distribution topology. In the following we describe all the protocol operations.

1) Peer join: The join procedure allows a new peer to enter into the overlay network. The same operation can be performed also by a peer that is already connected but whose video continuity is not satisfactory. When a peer P wants to join the network it first retrieves the address of the tracker and sends it a message containing its available slots $S(P)$. The tracker can assign P to a cluster served by a root source (level 1 cluster) or it can provide P with the address of a level 1 cluster-head (responsible of a cluster at level 2); the level 1 cluster-head is then contacted to recursively repeat the join operation. Reiterating this procedure, P can be joined at any level of the network. In other words, P follows a path along the control tree until it finds a cluster that can host it.

Acceptance of a new peer in a cluster C is done by the corresponding cluster-head. If $|\mathit{members}(C)| < N$, then it must guarantee that (1) holds, i.e., $S(P) \geq DL|\mathit{members}(C)| - \sum_{P' \in C}(S(P') - S_E(P'))$. In the case of a full cluster P is accepted if and only if $S(P) > \alpha \min_{P' \in C} S(P')$. In such a case the worst peer, i.e. $\arg\min_{P' \in C} S(P')$, is substituted by the new one. The worst peer uses a simplified join procedure and it is demoted to a higher level cluster. Parameter α must be greater than 1 to avoid frequent peer demotions. On the other hand, large values for α would result in a very strict criterion for the acceptance of a new peer. This cluster admission policy is introduced to ensure that more altruistic peers are placed closer to the root source to enjoy lower delays and higher stability in response to their higher contribution to the overlay.

2) Peer leave: A peer P can leave the network at any time notifying the following peers:

- the other members of $cluster(P)$;
- the sources of P, i.e., peers in $sources(cluster(P))$;
- the members of the cluster it serves (if P is a source);
- the other sources of the cluster it serves (if P is a source);

When departure of a peer is silent, e.g., due to a software or a hardware crash, its neighbors can infer it with some delay from the lack of periodic keep-alive messages. Keep-alive messages are sent from members of a cluster to their cluster-head. This type of message carries information about the peer free slots and information on its subtree (when the reporting peer is a cluster-head itself). The cluster-head runs a timeout for each of the members of its cluster; receipt of this message from a peer causes the reset of the corresponding timeout. The expiration of this timeout is equivalent to the reception of a quit message, so the cluster-head removes the peer from the list of members of that cluster and informs other members and sources of P about that. Keep-alive messages are also exchanged among the sources of a cluster (one of them is of course the cluster-head) to detect silent departures of sources (and cluster-heads) from the network. This allows all sources of a cluster to be informed about departure of P and enables the cluster-head to notify all the members about P absence. Such messages are piggybacked along with the control information required for peer bandwidth management described in Sect. II-C5.

After a departure several actions take place:

- all other members in its cluster and all its sources will stop sending packets to P;
- when a cluster-head detects that the leaving peer is one of the sources of its cluster it starts the *repair* process. The cluster-head sends a message to its father containing its IP address, the level of the cluster to be repaired and the missing stripe (d, l). If one of its children has enough upload bandwidth to replace the source for the missing stripe, then the repairing procedure is successful. If there is a children, that is serving as a cluster-head of a sub-tree with spare resources (gathered thorough previously mentioned periodic reports) the repair request is forwarded downward in the sub-tree. Otherwise, the repair message is forwarded upward to a lower level father, thus following a path along the control tree looking for a cluster-head able to attempt the repairing process. To avoid cyclical paths the repair message changes direction along the control tree only once. Moreover, the selected source must guarantee the service constraints depending on the cluster level and requested stripe. Since this process is not guaranteed to be successful it can be repeated after the expiration of a timeout. It is worth pointing out that in presence of a limited number of available slots a higher priority is given to the repair request issued by the lowest level clusters;
- if P is a cluster-head, another source will be elected cluster-head. If the sources of a cluster detect the silent departure of the cluster-head a successor is univocally self-elected on the basis of a priority policy based on the values of the stripe identifier. In particular, the cluster source forwarding the stripe with the lowest identifiers (d, l) is the one taking the lead.

If massive departures occur and all the sources of a cluster stop serving it the cluster turns to be disconnected from the network; in this unlikely case, the cluster peers reconnect by repeating the join procedure.

3) New cluster creation: A new cluster C' can be created below an existing cluster C, i.e. $\textit{level}(C') = \textit{level}(C)+1$; this decision is taken by the cluster-head, that appoints some of its children as sources of C'. The control information required for this operation is local to two levels of the control topology. To increase stability, the set $\textit{sources}(C')$ is not filled entirely using $\textit{members}(C)$. The remaining sources are gathered according to the repair procedure so as to minimize the overlap between the control and data topologies, thus limiting the probability of cascaded clusters' failures. C' will be initially an empty cluster to be populated by successive join requests.

The decision to create a new cluster C' must be taken only when the current clusters are almost full and it is likely to have enough upload resources to support it. This conditions guarantee that the control and distribution trees are kept as compact as possible to limit the communication delays and that the new clusters are well connected to the rest of the overlay. Going into details, the cluster is created if the following conditions are jointly satisfied:

- $\textit{members}(C) > \beta_1 N$, with $\beta_1 < 1$ a suitable constant, i.e. the number of $\textit{members}(C)$ is close to the maximum value N;
- $\sum_{C' \in \textit{headed}(C)} |\textit{members}(C')| > \beta_2 N |\textit{headed}(C)|$, with $\beta_2 < 1$ a suitable constant, i.e. the clusters headed by $\textit{members}(C)$ are well populated; the quantities $|\textit{members}(C')|$ are sent as control information by the cluster-heads of the next level and represent the only information that is not directly available at the cluster-head of C.
- $\sum_{P \in C} S_F(P) > DL\left(\gamma_1 + \gamma_2 |\textit{headed}(C)|\right)$, i.e., $\textit{members}(C)$ have enough free slots to support C'. In particular, the required number of free slots is made proportional to the number of controlled clusters.

4) Peers promotion and demotion: As already mentioned peers are not constrained to a given cluster but can move in response to network dynamics. Such migrations are implemented by means of a simplified join procedure. In particular, the cluster-head of a full cluster can move the worst peer, i.e., the one with the minimum contribution in slots, to a higher level cluster as far as a better peer joins its cluster. In the opposite case, a cluster-head that notes that the population of cluster C is falling below a certain threshold has means to select the best peers at level larger than $\textit{level}(C)$ by exploiting the periodic information sent by its children. As a consequence, it is possible to move the best peers of the higher levels to the cluster C. This approach keeps the overlay compact allowing to close the furthest clusters when the network population decreases.

As a last resource a peer is allowed to disconnect and reconnect in a new cluster if its quality of service is not satisfactory or because of the contemporary departure of all of its sources.

5) Upload bandwidth management: The management of the cluster upload bandwidth is done by the cluster-head. This is achieved on the basis of the periodic keep alive messages sent by its children, which, for each P, contain $S(P)$ and $S_E(P)$; clearly, only peers that are cluster sources have $S_E(P) \neq 0$. The cluster head objective is the allocation of the slots $S_I(P)$ for intra cluster streaming under the constraint that $S_F(P) = S(P) - S_E(P) - S_I(P) \geq 0$. The overall upload bandwidth required by a cluster C amounts to $D(C)L\left(|\textit{members}(C)| - 1\right)$ slots, where $D(C) \leq D$ represents the number of descriptions that can be supported by C. In fact, especially in leaf clusters that may host peers with limited resources, the churning could limit the slots availability for a certain period of time, thus forcing the cluster-head to reduce the streaming rate dropping some descriptions. If cluster peers were homogeneous in terms of $S(P)$, the optimal allocation would be $S_I(P) \approx D(C)L = \bar{S}_I$. In practice, peers are not homogeneous making this simple allocation sub-optimal. Moreover, within TURINstream we take into account the presence of MDC video coding so as to improve the performance in terms of the user experience avoiding, if possible, that the same peer relays more than one description. To this end the cluster-head performs slots allocation at a finer level of granularity, fixing the number of slots per peer P and description d: $S_I(P, d)$. It follows that $S_I(P) = \sum_d S_I(P, d)$ and that the number of slots allocated to the description d

in the cluster is given by $S_d = \sum_{P \in C} S_I(P,d)$. Therefore, the slots allocation sought by the cluster head is such that $S_d \geq L\left(|\mathit{members}(C)|-1\right), \forall d < D(C)$ and $S_F(P) \geq 0, \forall P \in \mathit{members}(C)$. Moreover, if possible the following constraint is enforced: $\exists! d : S_I(P,d) \neq 0, \forall P \in \mathit{members}(C)$ which amounts to preventing P to forward more than one description.

The allocation is refined dynamically starting from the value $S_I(P) = 0$, that is assumed when a new peer P joins the cluster. The algorithm used in TURINstream aims at achieving a balanced slots allocation $S_I(P)$. Every time a peer enters or leaves the cluster, the cluster head performs the following steps:

- Update $D(C)$ by selecting the maximum number of description satisfying (1), given the slots available in C. Then compute the average peer contribution $\bar{S}_I = D(C)L$.
- $\forall d : S_d > L\left(|\mathit{members}(C)|-1\right)$, remove slots allocated in excess (starting from peers holding more descriptions).
- $\forall P \in \mathit{members}(C)$ having $S_I(P,d) \neq 0$ for more than one d, keep the allocation only for description $d' = \arg\max_d S_I(P,d)$ and set $S_I(P,d) = 0, \forall d \neq d'$.
- $\forall P \in \mathit{members}(C)$ having $S_I(P) > \bar{S}_I$, $S_I(P) - \bar{S}_I$ slots are released.
- $\forall d : S_d < L\left(|\mathit{members}(C)|-1\right)$, find additional resources from the free slots of peers according to the following priorities:
  1) find every P such that $S_I(P) = 0$ and increment $S_I(P,d)$ by using some or all of its free slots $S_F(P)$ without exceeding $\bar{S}_I$;
  2) find every P such that $S_I(P) < \bar{S}_I$ and $S_I(P,d) \neq 0$ and increment $S_I(P,d)$ while keeping $S_I(P,d) \leq \bar{S}_I$;
  3) find every P such that $S_I(P) > \bar{S}_I$ and $S_I(P,d) \neq 0$ and increment $S_I(P,d)$;
  4) find every P such that $S_I(P) < \bar{S}_I$ and $S_I(P,d) = 0$ and increment $S_I(P,d)$ without exceeding $\bar{S}_I$;
  5) find every other peer P and increment $S_I(P,d)$.

Given the selected $S_I(P)$, the values of $S_F(P)$ of each children are known and can be used by the cluster-head to drive the repair and new cluster creation procedures described above. Finally, the values of $S_I(P)$ are shared with all the other cluster sources that must perform the actual forwarding of the video data according to such slots allocation. It is worth pointing out that the slots allocation is time dependent because of the dynamic of the cluster population and the resources they can provide. Let us make this dependence explicit by using $S_I(P,d,t)$ and $|\mathit{members}(C,t)|$ to represent the slots allocation and the population of cluster C at a given time instant t. At time t each source can evaluate the average bandwidth utilization of a peer P for a given description d as:

$$\rho(P,d,t) = \sum_{\tau=t_0(P)}^{t} \frac{\mathit{Bits}(P,d,\tau)\left(|\mathit{members}(C,\tau)|-1\right)}{\left(t-t_0(P)\right) S_I(P,d,\tau) r}$$

where $t_0(P)$ is the time when P has joined C and $\mathit{Bits}(P,d,\tau)$ is the amount of bits of the description d pushed toward P at time τ. The utilization $\rho(P,d,t)$ is defined as the average ratio between the bandwidth spent by P to forward video packets within C, measured by the source according to the size of the transmitted packets $\mathit{Bits}(P,d,\tau)$, and the bandwidth $S_I(P,d,\tau)r$ allocated by the cluster-head. At time t each source selects as destination of the next video packet of description d the peer $P(d,t)$ exhibiting the lowest utilization:

$$P(d,t) = \arg\min_{P' \in C} \rho(P',d,t).$$

III. Protocol simulator

The first step towards the development of the TURINstream prototype has been the design of an overlay simulator to aid the development of the algorithms and the optimization of the protocol parameters, e.g., number of stripes, number of descriptions, and number of peers per cluster. The simulator has been used to perform most of the optimization of the protocol before actually deploying it in a dynamic and distributed environment. In Sect. III-A we describe the simulator, in Sect. III-B the effect of the most critical protocol parameters is analyzed and finally in Sect. III-C the ability of TURINstream to scale to large networks is tested.

A. Simulator description

The behavior of the TURINstream protocol has been emulated by means of an event driven simulator. The simulator can handle networks of several thousands of peers and analyze their performance over time. The peers/clusters relations are modeled by a bipartite graph, composed by the objects Peers and Clusters. Bidirectional edges link the two classes and form a graph that models the TURINstream overlay topology. The most important simulated events are peer arrivals and departures (both notified and silent), cluster repairs and peer reconnection. Random latencies are used to accomplish each operation so as to simulate network delays. The simulator does not keep track of every message, so it can easily manage the dynamics of large networks for a long simulated time. In order to test the proposed protocol in a realistic scenario we adopted two different statistical models for peer arrival and sojourn times. In the first scenario, in the following referred to as departure process 1 (DP1), arrivals and departures are distributed according to negative exponential distributions, like in a M/M/∞ queue. In the second scenario a model where burst departures concentrate in short periods is considered (DP2). In particular, the simulated time interval is divided into cycles of one hour and at the end of every cycle 50% of the peers disconnect within a time interval of 100 s. The first model represents a behavior where the departures occur continuously, keeping the cluster repair system under pressure. The second is representative of mass peer departures e.g., after the end of a TV program, causing a major reorganization of the overlay. All simulations begin with an empty network, growing dynamically with peer arrivals; after an initial transitory, that is not considered in the computation of the performance indexes, its size became stable (in case of DP1) or begins its cyclical behavior (DP2). In both DP1 and DP2 20% of the departures are treated as silent to simulate unexpected peer crashes.

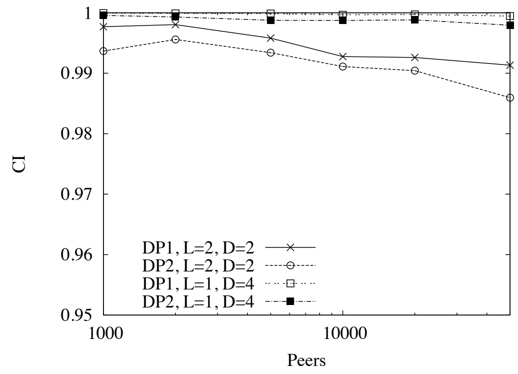


Fig. 3. CI versus the average overlay size with $N = 16$, $L = 2, 1$ and $D = 2, 4$ for DP1 and DP2 models.

Moreover, the simulator can handle peers with heterogeneous upload capacities. To obtain realistic results all the following experiments have been worked out assuming a typical upload distribution where the majority of the peer accesses the network though connections with limited upload bandwidth. The upload bandwidth distribution used in all the reported experiments is 384kbps (72%), 512kbps (24%) and 768kbps (4%). The video bitrate has been fixed to $R = 320$ kbps. We used a single root source node with upload capacity of $2R$ to initiate the video streaming.

The simulator goal is the computation of the percentage of time a peer P is getting the video. We did the pessimistic assumption that a description cannot reach a cluster if at least one of the paths from the root to the cluster transporting that description is broken somewhere. In other words, we assume that a video description is completely useless as far a as a single packet of such description is missing. This assumption does not hold for real video players where a limited number of packet losses can be mitigated e.g., by means of error concealment. At any time a peer P can be in $D + 2$ states $s = \{-1, 0, \dots, D\}$: it can be disconnected while trying to reconnect ($s = -1$) or it can belong to a cluster that receives $s = 0, \dots, D$ descriptions. During the simulation, the time $\mathrm{T}(P, s)$ spent by P in every possible state s is accumulated and the continuity index is evaluates as:

$$\mathrm{CI} = \frac{\sum_P \sum_{s=1}^{D} \mathrm{T}(P, s)}{\sum_p \sum_{s=-1}^{D} \mathrm{T}(P, s)}$$

Moreover, the simulator reports some topological indexes, namely the maximum and average height of the control tree.

B. Protocol parameters optimization

As a first result, the simulator has been used to select reasonable values for the following parameters: $\alpha = 1.3$ to manage the peer admission in a full cluster and $\beta_1 = 0.75$, $\beta_2 = 0.7$, $\gamma_1 = 2.5$, $\gamma_2 = 1.2$ to control the creation of new clusters. Such values have been chosen by studying their effects on the overlay topology and on the video service in terms of CI.

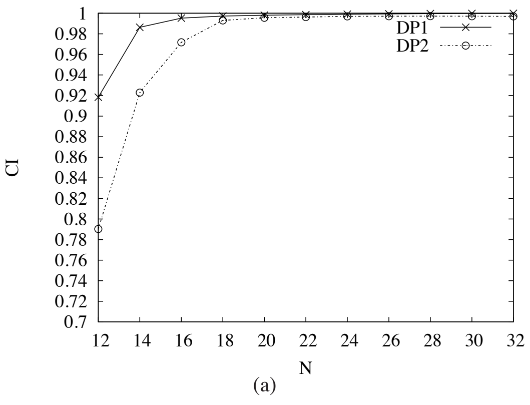
The most important parameters of TURINstream are the cluster size N and the number of stripes in which the video is fragmented, i.e. DL. In Fig. 2-(a) CI is reported versus N for an overlay of 1000 peers when fixing $D = 2$ and $L = 2$ for both DP1 and DP2 dynamic models. In Fig. 2-(b) we show the mean and maximum tree height versus N. As expected, larger values of N yield a more reliable and shorter control overlay network. On the other hand, using very large clusters is not feasible because of the increased upload required for a peer to share the video data with all his cluster-mates and the larger amount of control information to be managed by the cluster-head. It turns out that setting $N \geq 16$ is optimal from the point of view of both the overlay reliability and control tree height. Fig. 2 also points out that the mass departure scenario (DP2) is the most critical one yielding a lower CI and a higher maximum height. Nevertheless, such effect is particularly evident only when using small clusters.

C. Protocol scalability

The simulator has been used to test the behavior of the TURINstream overlay when the number of peers in the network increases up to the tens of thousands. The CI, the height of the control tree and the number of messages required for frequent protocol operations have been studied as a function of the average overlay size from 1000 to 50000 peers.

In Fig. 3 CI is shown for the cases $L = 2, D = 2$ and $L = 1, D = 4$ for DP1 and DP2 models. It can be noticed that TURINstream is able to scale to tens of thousands of users without a significant impact on the continuity index, that remains above 0.98 for both DP1 and DP2 peer dynamic models. Moreover, these latter experiments have been repeated for two configurations of the stripes settings. In the first case we use $L = 2, D = 2$ and in the second case $L = 1, D = 4$ so as to compare the performance of the protocol when the number of stripes is constrained to 4 but the number of video descriptions changes. Fig. 3 shows the improvements in terms of CI obtained using 4 descriptions (see square markers); clearly this advantage is due to the fact that when $D = 4$ the reception of a single stripe guarantees service continuity, even if with proportionally reduced video quality. Nevertheless, such gain can be achieved at the expense of higher video coding and decoding computational costs and a higher bitrate overhead with respect to standard video coding [7], [16]. The observation that TURINstream achieves a good performance with $D \leq 4$ is of paramount importance since most of the MDC video coding schemes available in the literature are designed for 2 or 4 descriptions [7], [16], [17], [18].

In Fig. 4-(a) the average and maximum heights of the tree of clusters build by TURINstream are reported versus the network size. It turns out that the average height of the control tree has logarithmic dependence on the overlay size (an $O(\log(\cdot))$ function is plotted for comparison). Clearly, a compact topology is able to guarantee low control and playback delays. Finally, in Fig. 4-(b) we show the average number of messages required for two TURINstream operations, namely the join and repair procedures. It turns out that a limited number of messages needs to be exchanged to accomplish the two tasks even for very large networks. The repair procedure, taking on average less than 5 control messages, turns out to be quite efficient and points out that the identification of



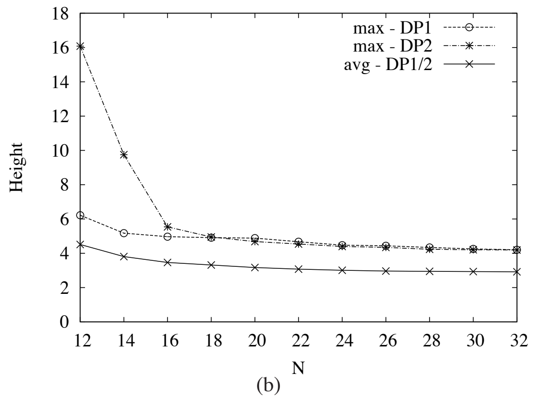
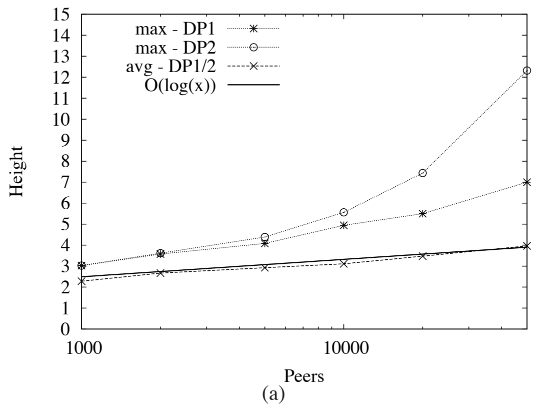


Fig. 2. CI (a) and control tree maximum and average height (b) versus the number of peers per cluster N, with $D = 2$, $L = 2$, overlay size of 1000, using DP1 and DP2 models.



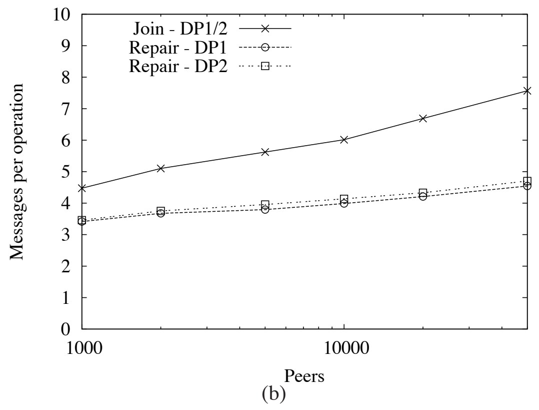


Fig. 4. Control tree maximum and average height (a) and average number of messages per operation (b) versus the average overlay size with $N = 16$, $L = 2$, $D = 2$ for DP1 and DP2 models.

additional resources requires only a local visit of the control tree. These latter achievements can be exploited to obtain the low protocol control overhead measured on the full prototype presented in the following section.

To conclude, for the prototype implementation and experimentation presented in Sect. IV, we selected a cluster size $N = 24$ and 4 stripes obtained by setting $D = 2$ and $L = 2$. The value of N has been selected as a compromise between reliability and upload bandwidth requirements for intra cluster communications. The number of stripes has been selected in order to accommodate the efficient MDC video coding technique recalled in the following that allow us to perform real time encoding of live video.

IV. Prototype development and experimental results

In the following we present details on the implementation of a prototype of TURINstream (Section IV-A) that we deployed and tested on PlanetLab (Section IV-B). We discuss results for the performance indexes we defined in Section IV-C that we obtained in several scenarios aimed at proving that TURINstream fulfills all the inspiring design goals. In particular, Section IV-D presents results when peers alternate between active and idle periods. In this case, performance of TURINstream are compared against those presented in [4]. Scalability properties of TURINstream are investigated in Section IV-E. Different arrival and departure patterns are considered in Section IV-F where most users join (flash crowd) and leave (mass departure) the application in a very short time. Finally, Section IV-G presents results in scenarios where the global amount of available upload resources (the so called resource index [19]) is rather limited, i.e., less than 1.

A. Prototype description

A complete prototype of the proposed P2P streaming protocol has been developed, using the C++ language. The application includes the client and the server implementing the protocol primitives described in this paper. The P2P protocol uses only UDP sockets so as to minimize transport delay and overhead. The server is equipped with a modified VideoLan [20] server in order to encode and stream the video according to the MDC algorithm presented in [7]. The client is able to receive the video descriptions, decode and play the video. The video descriptions are transported using RTP on top of UDP, whereas the video session and the decoder are set-up by means of SDP.

The proposed TURINstream is not constrained to the use of a particular MDC algorithm. Nevertheless the selection of the MDC video codec is important from the point of view of the added encoder/decoder complexity and coding overhead with respect to standard single description coding. The prototype has been worked out employing the MDC video technique [7], which is designed to create $D = 2$ video streams. According to this technique every video picture is fragmented into a number of slices; MDC coding yields 2 alternative representations of each slice, i.e., one packet for each description bitstream to be forwarded by the TURINstream protocol. This MDC technique has a number of desirable features: first of all it is compliant with the H.264/AVC standard [21], that achieves state of the art compression performance and allows one to use standard transport and signaling protocols. Moreover, as opposed to other solutions, the MDC coding overhead[1], expressed as the rate penalty with respect to standard video coding, can be controlled and freely adjusted according to the desired robustness/video quality trade-off. In other words, the MDC overhead can be selected so as to be hardly noticeable in absence of packet losses, while guaranteeing a more graceful degradation of the video quality when the P2P overlay is under stress. In the following experiments an MDC coding overhead ranging from 5% to 10% of the rate R has been used. Finally, the effort required on the decoder side to decode the descriptions is based on a simple RTP packet level selection algorithm, followed by a standard H.264/AVC player. The two descriptions are received as two separate RTP flows. The two streams are synchronized exploiting the RTP timestamps and merged into single AVC/H.264 RTP flow containing the best available representation per each slice; the output stream can be decoded by any standard player.

The TURINstream client and server applications are composed of four main modules:

- a thread that listens to the socket for incoming messages and processes them;
- an event list that is used to implement timeouts and periodic messaging;
- a video buffer where the incoming descriptions are synchronized and merged into a single video stream for playout according to [7];
- a queue that manages outgoing messages (control and video); in this queue smaller control packets are given higher priority so as to guarantee that critical operations, e.g. overlay repairs, are completed as quickly as possible.

B. Experimental testbed

The developed prototype has been initially validated by means of a local testbed; then it has been deployed using the PlanetLab platform to test the application in a realistic Internet scenario and to compare it versus other solutions.

We considered two scenarios:

- in the first setting we used our local PlanetLab node to host the video server (the root source) and a subset of other active PlanetLab nodes to run full clients, i.e., clients able to receive, mix and decode the video. We conducted experiments with a number of active PlanetLab nodes ranging from 50 to 200. We refer to this case as the *full scenario*.
- the second setting aims at analyzing the performance of TURINstream when scaling from hundreds to thousands of users. Since there are usually slightly more than 200 usable [2] PlanetLab hosts supporting our experiments the only solution is to allocate several peers per active host. Nevertheless, the bandwidth of a single host is not sufficient to support tens of concurrent peers, besides generating unrealistic and very correlated congestion patterns over the IP network. To overcome this limitation we considered a *lightweight scenario* where clients implement all the signaling procedures of the TURINstream protocol but video packets transmissions are emulated by sending only the packet headers. Clearly the peers bandwidth usage is computed by assuming the full size video packets. Peers push video packet headers in the network allowing to estimate all the performance indexes in absence of actual video streaming. The shortcoming of this approach is that IP network congestion may be underestimated. Still, the possibility to test the protocol scalability is a key point of our study and the obtained results are very accurate if one assumes that the underlying network infrastructure is not under stress. In fact, results obtained in the lightweight scenario have been successfully validated against results in the full scenario when the number of peers was equal to 200.

Moreover, all experiments have been performed by limiting the upload bandwidths; this was achieved by adding a software module between the application and transport layers, that drops packets if the upstream exceeds the selected limit. Such module is obtained controlling the rate at which messages are popped from the outgoing queue. This choice has two important motivations. First of all we are interested in simulating the system in a more realistic scenario comprising both residential ADSL and institutional users. More importantly, we want to make our results as reproducible as possible[3], even if we resort to the PlanetLab concurrent and open environment where resources cannot be guaranteed using a preemptive allocation.

According to the protocol optimization presented in Sect.III the prototype has been tested with the following protocol parameters: $\alpha = 1.3$, $\beta_1 = 0.75$, $\beta_2 = 0.7$, $\gamma_1 = 2.5$, $\gamma_2 = 1.2$, $N = 24$, $D = 2$ and $L = 2$. Moreover, the development of the full prototype and its testing in realistic scenarios has led to the selection of proper timeout values for the protocol operations. The most critical timeout values turned out to be the timeout for a cluster-head to receive a message from a child (T_{father}) and the timeout for periodic messages between sources ($T_{sources}$). The optimization of the protocol on the field has led to the selection of $T_{father} = 1$ s and $T_{sources} = 0.8$ s.

[1]All MDC coding techniques incur a rate penalty with respect to standard video coding. The coding overhead allows one to decode the descriptions independently and makes them mutually refinable.

[2]We use the term *usable* to denote a Planetlab node which was free of all the following problems: frequently unreachable due to downtimes and other issues, unable to reach (or be reached) by other Planetlab nodes, experienced a very bad connection quality, suffering from DNS problems, under very high load, varying SSH-keys, not enough disk space to write log files.

[3]unfortunately, bandwidth dynamics (congestion) cannot be controlled.

We also developed a console application to manage the joining and leaving of the peers and the collection of measurements. We conducted repeated experiments during September-November, 2009; results showed similar characteristics therefore we selected representative cases for the next discussion.

C. Performance indexes

The purpose of our experiment was to estimate some performance indexes to confirm that TURINstream satisfies our design goals. To assess the robustness of TURINstream we considered the continuity index (CI), that is expressed as the fraction of video packets that arrive to a peer before their playback deadline. Since TURINstream is designed to exploit MDC we consider that the video service is granted as far as at at least one out of the two representations of each video packet is received. To reflect this the continuity index (CI) is defined as the fraction of video slices for which at least one description has been received. In fact, in presence of MDC every video slice is split into two packets, one per each description, and a discontinuity occurs only if both packets are lost; if only one is received, the user does not perceives it as a discontinuity but as a degradation of the video quality.

Moreover, the developed prototype allowed us to measure the delays incurred by the distributed application. The protocol latency is measured by means of the following indexes:

- Connection delay t_C, defined as the time interval between the sending of the first join request message and the reception of the accept message.
- Startup delay t_S, defined as the interval between the first join message and the decoding of the first video frame; it includes connection time and buffering time. Hence, it is the time between the application starts and the time instant the user starts watching the video.
- Playback delay t_P, defined as the amount of time elapsed from the first transmission of a packet in the overlay by the server and its actual playback time experienced by the user.

Finally, the efficiency of TURINstream has been evaluated in terms of signaling overhead. Two types of overheads can be identified, namely those caused by the video packet headers and by messages carrying control information, respectively. The video packet headers are 14 bytes long, summing up to 28 bytes if one takes into account the UDP headers as well; this overhead is negligible and it is similar to the one yielded by standard streaming transport protocols such as RTP/UDP. Clearly, the major source of overhead is represented by signaling information exchanged to maintain the control and video streaming overlay topologies. Moreover, such contribution depends on the peers/network dynamics. To measure this latter, the overall amount of control traffic is logged by each peer. This allows us to measure the protocol overhead as the ratio between the average bitrate of the control information over the video bitrate.

D. On-off behavior

Performance of novel P2P video streaming architecture should be compared against those of other techniques. This important issue is nevertheless very difficult to deal with. Indeed, a thorough and fair comparison may not be accurate due to the difficulty of re-creating the same test environment [4]. For instance, while it would be theoretically possible to compare different architectures using the same video bitrate, root source upload bandwidth, peers upload bandwidth distribution, arrival (departure) pattern, number of peers in the overlay, it is surely impossible to reproduce the congestion and CPU load experienced by the PlanetLab nodes. Of course, this is true only if the prototypes implementations of other techniques were publicly available for experimentation. If this is not the case then it is only possible to provide a comparison where the maximum number of system parameters is matched to obtain hints on the relative performance of two competing architectures.

TABLE II
PARAMETERS OF THE TEST ENVIRONMENTS SHARED WITH [4]

System parameter	Value
R	500 Kbps
Number of peers	50 to 200
Arrival (departure)	Exponentially distributed (average 50s to 200s)

We chose to compare the performance of TURINstream against those of Coolstreaming as presented in [4]. In particular, we evaluated the average CI and control overhead performance indexes where we matched the video bitrate, the peer arrival (departure) pattern, the number of peers in the overlay (Table II shows the parameters of the test environments that match those used in [4]). We were not able to obtain information on the root source upload bandwidth and on the peers upload bandwidth distribution. In [4] the CI and the overhead have been measured using 4 connections per peer (including the root source); therefore it seems reasonable to assume that the root source upload bandwidth has been fixed to $4R$. On the other hand, no limitations seems to have been applied to the peers upload bandwidth.

To show that TURINstream is capable of providing performance at least as good as those of Coolstreaming we fixed our root upload bandwidth to only $2R$ (half of what has been presumably used in [4]). This choice is representative of a scenario where a user wants to stream its video content without resorting to a dedicated networking infrastructure. It follows that results could be significantly improved if a better connected video provider was considered, e.g., representative of a commercial video distribution system. In fact, by increasing the number of level 1 clusters the continuity index would increase and delays would shorten for a larger number of peers. The control overhead would practically be unaltered.

We also imposed a cap on the peers upload bandwidth by using the upload bandwidths of the participating peers according to the distribution shown in Tab. III. The second column represents upload values distributed according to bandwidth available to a majority of home ADSL users and a limited percentage of institutional/business users with high capacity. The resource index is equal to 1.18.

[4]We do not consider architectures whose performance have been evaluated only through simulations.

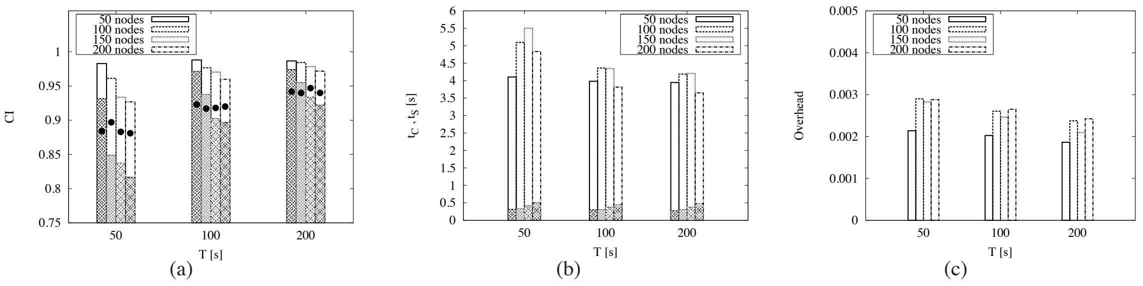


Fig. 5. TURINstream average CI (empty bars) and the ratio between the average number of received descriptions and D (grey bars) along with experimental results from CoolStreaming [4] (black dots) (a), t_C (gray bars), t_S (b) and protocol overhead (c) for networks with 50,100,150,200 peers as a function of the average ON/OFF period T (full scenario).

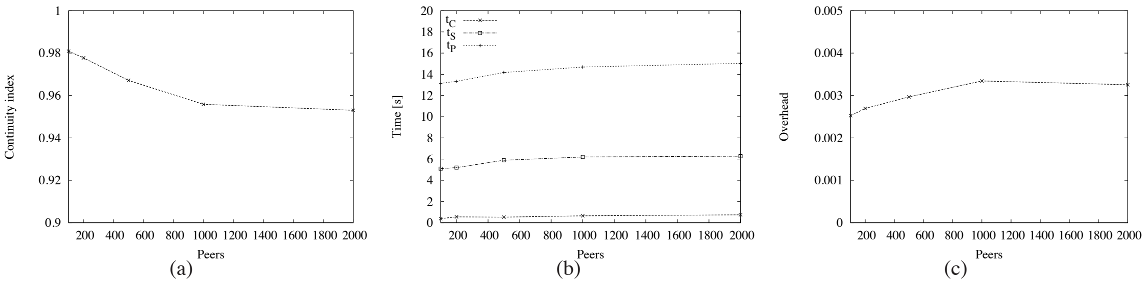


Fig. 6. Performance indexes as a function of the number of peers (lightweight scenario): average CI (a), t_C, t_S, and t_P (b) and overhead (c).

TABLE III
LIMITED UPLOAD BANDWIDTH DISTRIBUTION

%	TURINstream upload bandwidth [kbps]
8	397
64	530
12	662
12	795
4	1060

Moreover, we decided that peers' departures are notified with probability 0.95 while they are silent with probability 0.05 to simulate crashes. Silent departures are obtained by suppressing all quit messages. This is another feature that makes our experimental conditions more adverse than those in [4].

All our experiments lasted for one hour and performance indexes are computed over all the ON periods. Fig. 5(a) shows the TURINstream performance in terms of CI as a function of the average ON/OFF period T for different network sizes. The four bars in the histograms show the CI obtained with different overlay sizes of 50,100,150 and 200 nodes; the dash level in each bar shows the ratio between the average number of received descriptions and D and it is representative of the averaged received video quality; the black dots are experimental values of CI reported in [4]. Fig. 5(a) shows that TURINstream achieves a high CI by profitably exploiting MDC. In particular, it can be noticed that for $T = 100$ and $T = 200$ s TURINstream yields $CI \geq 0.97$ with an average number of received descriptions very close to D. In presence of a higher churn, e.g. $T = 50$ s, the TURINstream architecture still yields CI larger than 0.9 at the expense of a slight reduction in the number of received descriptions. The results from [4] are reported in order to compare TURINstream performance with that of the first popular mesh based streaming system; nevertheless, it must be recalled that in [4] no limitation on the peer upload bandwidths were considered and the root source upload bandwidth was likely to be double with respect to that used for TURINstream .

TURINstream exhibits limited latencies, taking on average less the 500 ms to complete the join procedure as shown in Fig. 5(b) (filled bars for t_C) in all cases. In the same figure a low startup delay t_S of about 5 s is reported. The playback delay t_P is about 12 s in all cases. We also computed the CDF for t_S in the case $T = 100$ s and 100 nodes (the other cases are qualitatively very similar); we observed that the 95^{th} percentile is equal to 6.1 s and the 99^{th} percentile is equal to 6.6 s.

Finally, the proposed design is very efficient also in terms of signaling overhead which is kept below 0.003 as shown in Fig. 5(c) which is one order of magnitude smaller than what reported in [4]. The main contribution to the overhead is given by periodic keep-alive messages. There are 3 kind of such messages:

- every peer sends a message to its cluster-head every T_{father} seconds containing information for bandwidth management;
- every source sends a keep-alive message to the cluster-head every $T_{sources}$ seconds;
- the cluster-head communicates to the sources the slots allocation every $T_{sources}$ seconds.

An approximation of the average overhead per peer due to the keep-alive messages is given by:

$$\frac{(472 + 8N)}{T_{father}} + \frac{(736 + 8DN)(DL - 1)}{(T_{sources}N)}[bps] \qquad (2)$$

where the first term takes into account the messages from a peer to its cluster-head, while the second one is the contribution of the cluster head. This latter represents the cost of sending the DN values of $S_I(P,d)$ to the $(DL-1)$ other sources. The constant values represent the size of control packets, including the cost of UDP headers, that do not depend on any system parameter. As a sanity check we computed (2) when substituting the protocol parameters values reported in Sect. IV-B: one obtains 839 bps, which amounts to about the 0.2 % of the video bitrate. This estimate confirms the results shown in Fig. 5(c). The discrepancy is due the fact that (2) neglects the contributions of control information exchanged for protocol operations, e.g., join, leave, cluster repair, etc.

To summarize TURINstream is able to obtain low delays, as those previously reported for tree based topologies, and at the same time robustness to peer churning and very low control overhead. The comparison against the mesh based architecture Coolstreaming reveals that TURINstream performance are at least as good for CI, while it outperforms Coolstreaming for both control overhead and delays.

E. Scalability

To test system composed of a larger peer population we considered three hours long experiments in the lightweight scenario where peers' inter-arrival and permanence times are generated according to an exponential distribution, i.e., the amount of peers in the system behaves like a M/M/∞ queue. The exponential distribution parameters have been set to obtain an average sojourn time $T=120$ s and steady-state average overlay sizes ranging from 100 to 2000 peers.

Fig. 6(a)-6(c) show the performance indexes versus the average overlay size up to 2000 peers. It can be noticed that TURINstream is able to scale to thousands of peers without significant impairment in terms of CI (Fig. 6(a)), delays (Fig. 6(b)) and signaling overhead (Fig. 6(c)).

F. Mass behavior

We also tested the performance of TURINstream in a more challenging setting where a large number of peers joins the application in a short amount of time (flash crowd). To this end, we considered a system of 1000 peers in the lightweight scenario. For flash crowds we considered rate of peer arrivals ranging from 50 to 200 peers/s while for mass departure we forced a percentage ranging from 10% to 40% of the peers to leave the application in a time interval of 5 s (these departures are silent with probability 0.05). The behavior of CI in presence of a flash crowd starting at time instant 0 is shown in Fig. 7-(a). It can be noted that the application is robust under sudden massive arrivals. In fact, the initial unavoidable drop in CI is recovered in a few seconds. When arrivals occur on average every 5 ms we note that after 30 seconds the average CI is above 0.9 and stabilizes to slightly less than 1 because no departures occur thereafter. In this case t_C and t_S slightly increase to 3 s and 8 s, respectively. These values must be compared with those experienced with 1000 peers in Fig. 6(b) where $t_C=500$ ms and $t_S=6$ s; it turns out that t_C is the most sensitive to flash crowds, whereas the impact on t_S is limited. Control overhead is only marginally affected. As a final comment, one has to keep in mind that these results have been obtained starting from an *empty* overlay; the presented results improve if the flash crowd arrivals occurred on an already well populated network.

TABLE IV
LIMITED UPLOAD BANDWIDTH DISTRIBUTION

class id	upload bandwidth [kbps]	S1	S2	S3	S4
1	230	10%	20%	30%	40%
2	345	10%	20%	30%	40%
3	460	70%	50%	30%	10%
4	690	5%	5%	5%	5%
5	920	5%	5%	5%	5%
	Resource Index	1.09	1.01	0.93	0.84

The effect of massive departures on CI is reported in Fig. 7(b); this latter clearly shows that the TURINstream is able to efficiently cope with the sudden reduction of the resources experienced by the system when many peers leave almost at the same time. In particular, it can be noted that the continuity index never drops below 0.8 even when 40% of the peers leave within 5 seconds. The delay of about 5 s with respect to the start of the mass departure event at time 0 is due to the presence of video buffering. In presence of mass departures delay and overhead values show negligible variations.

G. Quality of service differentiation under limited resources

To test the ability of TURINstream to provide different quality of service to peers sharing more resources, we considered scenarios where the resource index is significantly lower than what we used in the previous sections, i.e., 1.18. In particular, we considered a video bitrate $R=420$ Kbps, a root server upload bandwidth equal to $4R$ and peers upload bandwidth distributed according to five classes in four scenarios as summarized in Table IV. We considered a lightweight scenario where the average sojourn time is equal to 360 s while the inter-arrival time is equal to 720 ms and 360 ms to obtain an average overlay size equal to 500 and 1000 peers, respectively. Experiments lasted for three hours; we computed the average CI, the average number of descriptions Q, and the startup delay t_S obtained by peers in each bandwidth class for each of the four scenarios. Results presented in Table V prove that TURINstream allows altruistic peers to enjoy very high CI and almost full quality video reception. Indeed, this goal is obtained even in scenarios with very scarce available resources (S3 and S4) and it is maintained as the number of peers doubles. The more peers act selfishly the more they are penalized; it can be noted that peers in class 1 (that contribute no more than half the video bitrate) enjoy low CI and Q while the quality of service experienced by class 2 peers (that contribute for three quarters the video bitrate) are consistently better. It can also be noted that the startup delay t_S is lower for more cooperating peers: this result is due to the fact that TURINstream is able to accommodate more altruistic peers into the top levels of the overlay through promotion and demotion operations.

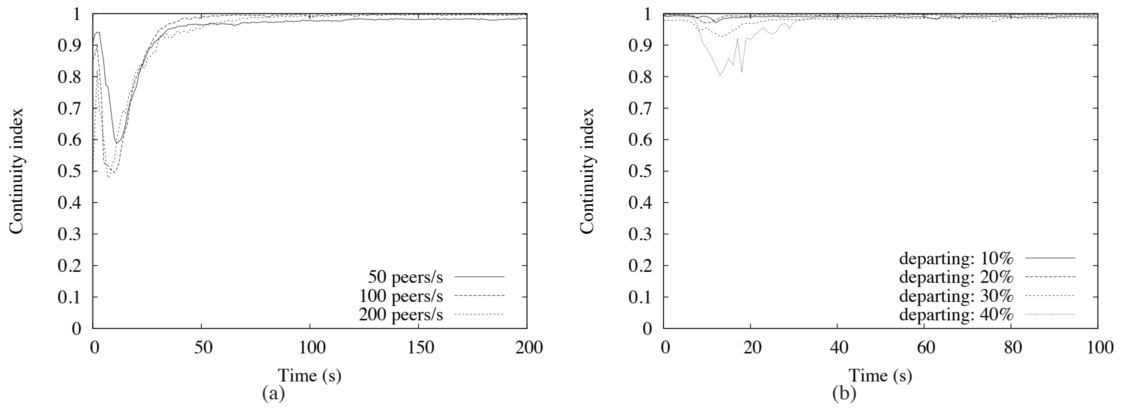


Fig. 7. TURINstream continuity index for flash crowd arrivals (a), and mass departures (b).

TABLE V
LIMITED UPLOAD BANDWIDTH DISTRIBUTION

	S1						S2					
	500 peers			1000 peers			500 peers			1000 peers		
	CI	Q	t_S	CI	Q	t_S	CI	Q	t_S	CI	Q	t_S
class 1	64.2	0.84	6.26	62.9	0.83	6.50	59.5	0.68	6.34	58.7	0.67	6.39
class 2	83.1	1.14	5.96	80.2	1.10	6.05	77.3	0.91	6.19	74.8	0.87	6.23
class 3	98.4	1.83	5.07	98.1	1.81	5.13	97.5	1.76	5.19	97.5	1.74	5.09
class 4	99.6	1.95	4.36	99.1	1.93	4.47	99.2	1.92	4.57	99.2	1.92	4.47
class 5	99.6	1.96	4.34	99.2	1.94	4.41	99.4	1.95	4.57	99.3	1.92	4.36
	S3						S4					
	500 peers			1000 peers			500 peers			1000 peers		
	CI	Q	t_S	CI	Q	t_S	CI	Q	t_S	CI	Q	t_S
class 1	61.9	0.68	6.32	52.8	0.58	6.42	49.4	0.53	6.51	53.9	0.58	6.59
class 2	73.7	0.81	6.13	67.6	0.75	6.35	66.4	0.74	6.19	67.8	0.73	6.33
class 3	98.5	1.85	4.82	95.3	1.66	5.34	98.1	1.80	4.88	95.4	1.69	5.06
class 4	99.5	1.95	4.42	98.3	1.86	4.68	99.0	1.87	4.64	98.4	1.87	4.56
class 5	99.5	1.95	4.36	99.2	1.92	4.55	99.0	1.88	4.61	99.1	1.92	4.48

V. RELATED WORKS

The TURINstream architecture encompasses two key elements, namely the MDC video distribution on a multi-tree topology and the organization of the peers in clusters. In the past many multi-tree solutions have been proposed where the node of the trees are the peers as opposed to the TURINstream clusters. Here we mention CoopNet [1] and Splitstream [2] that are two well known approaches. In CoopNet a set of random random trees is built by using a centralized approach. All the peers joining and leaving the overlay exploit a resourceful server node that coordinates the overlay construction and optimization. Because of the absence of a distributed and local algorithm for the construction of the trees the overlay reorganization in response to peer crashes or bandwidth fluctuation can be costly. Moreover, CoopNet exploits a high number of video descriptions (up to 16) to make the offered service reliable. On the other hand, SplitStream represents a viable distributed approach to the construction of the multi-tree; in this latter case the use of *interior-node-disjoint* trees is proposed, where each peer is an interior node in at most one tree. This solution limits the impact of the departure of a peer to a single tree. Nevertheless, it has the negative drawback that every peer is a leaf node in the distribution of $D-1$ out of the D descriptions. In presence of churn the quality of service in terms of both delays and continuity decreases with the height of the tree. This fact heavily limits the overall quality offered to each peer.

A few peer-to-peer architectures have introduced the concept of clustering in recent years as well. The Zigzag [22] protocol, which is an enhanced version of NICE [23], builds a single clustered multicast tree. As opposed to TURINstream in ZigZag all the peers belong to a cluster at level 0, whose cluster-head belongs to a cluster at level 1; then level 1 cluster-head belong also to a cluster in level 2 and so on. As a consequence, in [22] a top level cluster-head belongs to all the clusters; on the other hand, in TURINstream each cluster-head has only a local knowledge of the overlay making it more robust to peer churn. In particular, TURINstream repair operations are quicker since the topology is more flexible and does not require split and merging of clusters as in ZigZag. Moreover, TURINstream uses a multi-tree for content distribution whereas ZigZag employs a single multicast tree. Finally, in [22] simulation based results without the exploitation of MDC have been reported. [24] presents another layered cluster architecture very similar to ZigZag from the point of view of the overlay topology. The focus of this work is on optimal exploitation of peer bandwidth whereas the proposed architecture, as most of the structured topologies, is not resilient to peer churn and departures. In particular, in dHCPS the cluster-heads represent a single point of failure. Furthermore, experimental results in [24] have been worked out for small and stable systems (80 peers on PlanetLab) and considering a video server equipped with 3.2 Mbps upload bandwidth capacity; actual scalability and resilience are thus a concern for this architecture.

Another goal pursued in this work is service differentiation based on peers contribution so as to promote altruistic behaviors. In [25] a centralized solution that organizes peer in a direct acyclic graph and adapts the video streaming rate based on the peer contributions is proposed and analyzed by simulation. The system proposed in [25] requires a parent node to transcode incoming stream to serve its children at different rates. Transcoding is more flexible in terms of the achievable rates but is highly demanding in terms of computational resources with respect to the use of MDC, where peers can simply drop some description to shape the rate.

VI. Conclusions

In this paper we presented TURINstream, an architecture for P2P based video streaming. It exploits separation of data and control overlays, cluster based organization, MDC video coding, and tree-based control overlay to achieve a high degree of robustness, low connection, startup and playback delays, high scalability, low control overhead, and differentiated quality of services for different classes of users.

TURINstream design has been optimized through an event driven overlay simulator able to scale up to tens of thousands of users. A complete prototype of TURINstream has been developed, deployed and tested on PlanetLab under varying degree of peer churn, flash crowd arrivals, sudden massive departures, and upload bandwidth limitation. Moreover, the prototype has been equipped with state of the art MDC video coding allowing us to test the streaming protocol under realistic video traffic. The main finding is that TURINstream fulfills our initial design goals and proved to be competitive with other architectures.

Future efforts will be devoted to devise more efficient exploitation of all the available upload bandwidth of participating users as well as to cope with the delay in adaptation to bandwidth fluctuations. Moreover, network aware algorithms will be considered for optimal cluster formation.

Acknowledgment

The authors would like to thank the anonymous reviwers for their valuable comments and suggestions that contributed to improve the paper. This work has been partially supported by the FP7 COAST (FP7-ICT-248036) project, funded by the European Community.

References

[1] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *ACM NOSSDAV*, May 2002, pp. 177–186.

[2] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," in *ACM SOSP*, Oct. 2003, pp. 298–313.

[3] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Resilient peer-to-peer Streaming," in *IEEE ICNP*, Nov. 2003, pp. 16–27.

[4] X. Zhang, J. Liu, B. Li, and Y. S. P. Yum, "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming," in *IEEE INFOCOM*, Mar. 2005, pp. 2102–2111.

[5] X. Hei, Y. Liu, and K. W. Ross, "IPTV over P2P Streaming Networks: The Mesh-Pull Approach," *IEEE Communications Magazine*, vol. 46, no. 2, pp. 86 –92, Feb. 2008.

[6] V. Vaishampayan, "Design of multiple description scalar quantizers," *IEEE Trans. on Information Theory*, vol. 39, no. 3, pp. 821–834, May 1993.

[7] T. Tillo, M. Grangetto, and G. Olmo, "Redundant slice optimal allocation for H.264 multiple description coding," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 18, no. 1, pp. 59–70, Jan. 2008.

[8] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree: A comparative study of live p2p streaming approaches," in *IEEE INFOCOM*, 2007.

[9] Z. Liu, Y. Shen, S. Panwar, K. Ross, and Y. Wang, "P2P video live streaming with MDC: Providing incentives for redistribution," in *ICME*, Jul. 2007, pp. 48–51.

[10] N. Magharei and R. Rejaie, "Prime: peer-to-peer receiver-driven mesh-based streaming," *IEEE/ACM Trans. Netw.*, vol. 17, no. 4, pp. 1052–1065, 2009.

[11] M. Zhang, Q. Zhang, L. Sun, and S. Yang, "Understanding the Power of Pull-Based Streaming: Can We Do Better?" *In IEEE Journal on Selected Areas in Communications*, vol. 25, no. 8, pp. 1678–1694, Dec. 2007.

[12] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg, "Epidemic live streaming: optimal performance trade-offs," in *SIGMETRICS*, Jun. 2008, pp. 325–336.

[13] *RFC 3984, RTP Payload Format for H.264 Video*, Feb. 2005.

[14] *RFC 2326, Real Time Streaming Protocol (RTSP)*, Apr. 1998.

[15] *RFC 5583,Signaling Media Decoding Dependency in the Session Description Protocol (SDP)*, Jul. 2009.

[16] R. Bernardini, M. Durigon, R. Rinaldo, L. Celetto, and A. Vitali, "Polyphase spatial subsampling multiple description coding of video streams with H.264," in *IEEE ICIP*, Oct. 2004, pp. 3213–3216.

[17] E. Akyol, A. Tekalp, and M. Civanlar, "A flexible multiple description coding framework for adaptive peer-to-peer video streaming," *IEEE Journal of Selected Topics in Signal Processing*, vol. 1, no. 2, pp. 231–245, Aug. 2007.

[18] C. Zhu and M. Liu, "Multiple description video coding based on hierarchical B pictures," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 19, no. 4, pp. 511–521, Apr. 2009.

[19] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The feasibility of supporting large-scale live streaming applications with dynamic application end-points," in *ACM SIGCOMM*, Feb. 2004, pp. 107–120.

[20] *VideoLAN project*. [Online]. Available: http://www.videolan.org/

[21] Joint Video Team JVT of ISO/IEC MPEG and ITU-T VCEG, *Intl. Standard of Joint Video Specification (ITU-T Rec. H.264, ISO/IEC 14496-10 AVC)*, Mar. 2003.

[22] D. A. Tran, K. Hua, and T. Do, "ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming," in *IEEE INFOCOM*, Mar. 2003, pp. 1283–1292.

[23] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *ACM SIGCOMM*, Aug. 2002, pp. 205–217.

[24] Y. Guo, C. Liang, and Y. Liu, "dHCPS: decentralized hierarchically clustered p2p video streaming," in *ACM CIVR*, Jul. 2008, pp. 655–662.

[25] W. Ooi, "Dagster: Contributor aware end-host multicast for media streaming in heterogeneous environment," in *MMCN*, Jan. 2005, pp. 77–90.

Andrea Magnetto was born in Rivoli, Italy, in 1984. He received his degree in Computer Science in 2006 from the University of Torino. His main research interests include network modeling/simulation, with particular interest in peer-to-peer streaming

Rossano Gaeta received his Laurea and Ph.D. degrees in Computer Science from the University of Torino, Italy, in 1992 and 1997, respectively. He is currently Associate Professor at the Computer Science Department of the University of Torino. He has been recipient of the Best Paper award at the 14-th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2006) and at the 26th International Symposium on Computer Performance, Modeling, Measurements, and Evaluation (PERFORMANCE 2007). His current research interests include the design and evaluation of peer-to-peer computing systems and the analysis of compressive sensing and coding techniques in distributed applications.

Marco Grangetto (S'99-M'03-SM'09) received the Electrical Engineering degree and the Ph.D. degree from the Politecnico di Torino, Turin, Italy, in 1999 and 2003, respectively. He is currently an Assistant Professor at the Computer Science Department, University of Torino. His research interests are in the fields of multimedia signal processing and networking. In particular, his expertise includes wavelets, image and video coding, data compression, video error concealment, error resilient video coding unequal error protection, and joint source channel coding. Dr. Grangetto was awarded the Premio Optime by Unione Industriale di Torino in September 2000, and a Fulbright grant in 2001 for a research period with the Department of Electrical and Computer Engineering, University of California at San Diego. He has participated in the ISO standardization activities on Part 11 of the JPEG 2000 standard. He has been a member of the Technical Program Committee for several international conferences, including the IEEE ICME, ICIP, ICASSP, and ISCAS.

Matteo Sereno (M'08) was born in Nocera Inferiore, Italy. He received his Laurea degree in Computer Science from the University of Salerno, Italy, in 1987, and his Ph.D. degree in Computer Science from the University of Torino, Italy, in 1992. He is currently Full Professor at the Computer Science Department of the University of Torino. His current research interests are in the area of performance evaluation of computer systems, communication networks, peer-to-peer systems, sensor networks, queuing networks and stochastic Petri net models.