Journal of Mobile Multimedia, Vol. 0, No. 0 (2009) 203–237
© Rinton Press

MESH-BASED P2P LIVE VIDEO STREAMING WITH STREAMCOMPLETE [a]

FEDERICO COVINO MASSIMO MECELLA
Dipartimento di Informatica e Sistemistica ANTONIO RUBERTI, SAPIENZA *Università di Roma*
via Ariosto 25, Roma, 00185, Italy
federico.covino@gmail.com, mecella@dis.uniroma1.it

Received July 21, 2007
Revised March 26, 2009

This paper introduces StreamComplete, a new architecture and prototype system for mesh-based P2P live video streaming; it realizes a new concept of overlay network's management as well as merges the best practices of tree- and mesh-based approaches. StreamComplete creates a dynamic overlay network that optimizes itself on the basis of local conditions of peers. We have extensively evaluated the performance and the behavior of StreamComplete over the PlanetLab. Our experiments demonstrate the ability of peers in managing the overlay network with autonomic behavior in case of network changes. Furthermore the system is scalable w.r.t. the cardinality of the overlay network, by requiring very few control traffic.

Keywords: Net-TV, peer-to-peer video streaming, mesh networks

Communicated by: D. Naniar

1 Introduction

1.1 Net-TV

The merging of a social phenomenon like Web 2.0 with multimedia communication technologies, and the availability of inexpensive broadband Internet connections for home-users, has recently made practical a large number of bandwidth-intensive applications, e.g., Network Television (Net-TV). This is a new communication media: each user, each single person, can broadcast his own video information to the world in real time and without charge. Net-TV examples are Joost, BabelGum, Mogulus, Tribler, Veoh, etc.: each of them achieves a television infrastructure on the basis of a computer network. The last commercial trends combine the classic TV in the living room with Net-TV applications. From the hardware point of view, it is possible to connect the classic TV to the IP network through a simple set-top-box, similar to a decoder: it represents the television's network adapter. Furthermore, to avoid any limitation of the interactivity in searching and displaying, it is possible to abandon the classic remote control to switch to the new 3D-mouses (e.g., see Figure 1).

The most common applications of Net-TV are: *(i)* for Public Administrations, services to citizens and promotion of tourism; *(ii)* for the so-called Third Sector, promotion of local

[a]The work has been partially supported by the EU Commission through the FP6 project WORKPAD and the FP7 project SM4ALL.

Fig. 1. Example of a 3D-mouse

events and demonstrations; *(iii)* for entertainment, film, television programmes and music channels; *(iv)* User Generated Content.

1.2 From IP Multicast to P2P-TV

For such a kind of applications, the IP multicast protocol might probably be the most efficient vehicle. But due to the practical issues of routers, this protocol has not been widely deployed in the wide-area network infrastructure. The IP multicast model extends the traditional best-effort unicast model with efficient multi-point packet transmissions. The key innovation is to ensure simultaneous dissemination of packets to a set of destinations traversing each link only once without duplication. This is achieved through the construction of a spanning tree across routers in networks. There are primarily two issues within the IP multicast framework:

- heterogeneity: each receiver differs in its computing capability and access bandwidth;
- dynamic: video playback quality varies in the best-effort Internet.

IP multicast encountered the following problems in the deployment:

- scalability: there are potentially a large number of multicast groups that must be managed in a large network;
- a requirement for coordination of dynamic spanning tree(s) construction at routers across different autonomous subnets, which often makes it practically infeasible;
- routers must maintain the state, which violates (stateless) principles and creates difficulty in the design of high-level functions such as error, flow, and congestion control.

[21] presented the key difficulties encountered in the native IP multicast and proposed moving the multicast functionality to the edge of the network. The construction of a multicast tree was performed using an overlay network, that is, a virtual topology over the unicast Internet. This demonstrated the feasibility of implementing multicast functions at the edges of the systems while keeping the core functionality of the Internet intact. The significance of such an approach is to enable efficient multicast service deployment without the requirement for infrastructure support [17].

Therefore, researchers resorted to application-level solutions, which build a peer-to-peer (P2P) overlay network out of unicast tunnels across cooperating end-users: the so-called P2P-TV.

For sake of completeness, we mention another important trend, which is to combine the mobile networks with the multimedia applications (including video streaming). This has been started by the introduction of the IP service on top of the Global System for Mobile Communications (GSM), then on top of the General Packet Radio Service (GPRS) and finally on top of the Universal Mobile Telecommunications System (UMTS). Also in this context, P2P architectures have been deeply studied in recent years and several mobile frameworks have been proposed [1, 2, 3]. Mobile P2P streaming can be achieved as well as an adaptation of the wired P2P streaming.

1.3 Outline and Contribution of the Paper

In this work, a new mesh-based P2P live video streaming system, called StreamComplete, is proposed and realized. It merges new algorithms with the current best practices of the tree- and mesh-based approaches in order to achieve a system capable of creating a dynamic overlay network that optimizes itself on the basis of the local conditions of peers. Particular attention was paid in order to reduce the time of connection, the delays and the traffic control overhead. StreamComplete is based on two distributed algorithms: *(i)* the Fast Top procedure that improves the general performance of the overlay intervening on the network topology; *(ii)* the Loop Check procedure that checks the presence of bounded loops in the mesh graph. The logical links between peers are managed on the basis of the peers' *state of health*: this parameter summarizes the condition of available resources in the peer and the quality of its overlay relationships. StreamComplete entrusts the video stream on top of the RTP/UDP protocol and the control traffic on top of the TCP protocol. Thanks to the StreamComplete's modularity, exchanging the video stream technology without involving the core engine of the application is very easy.

We have extensively evaluated StreamComplete's performance and behavior over almost all the nodes of the PlanetLab testbed[a]. The result demonstrates the ability of peers of joining the system in only few seconds. All the algorithms of network control and optimization generate almost zero additional traffic, when compared with data traffic. The network reacts very well in case of departing or high churn rate. The average node degree is low, in agreement with the small worlds property [15].

This paper is organized as follows: Section 2 compares with some related works, Section 3 presents the system, whereas Section 4 introduces and discuss the algorithms conceived for the system. Section 5 presents the experimental evaluation, and finally Section 6 concludes the paper.

2 Related Works

In this section we compare the main academic (i.e., Chunkyspread [5], CrossFlux [6], CoolStreaming [7], Prime [8, 9], LSONet [10], Rainbow [11], Tribler [14], GridMedia [19, 20]) and commercial (i.e., Joost [12], BabelGum [13]) proposals in order to identify similarities, differences, pros and cons.

2.1 Background

The key characteristics of a P2P-TV system is, first, the use of a pure or hybrid P2P approach and, second, the topological organisation of the overlay network. Hybrid P2P approach means

[a]PlanetLab – `http://www.planet-lab.org/` is a global research network, consisting of different nodes whose number ranges from 150 to 400 and over, ideally with a maximum of 800, that allows distributed experiments.

that there is also the use of a client/server architecture embedded into some points of the overlay network, usually dedicated to solve specific sub-problems of the service: this is the case of Joost, BabelGum, GridMedia and of all the BitTorrent-based systems, like Tribler.

Furthermore, there are two main possible architectures for the overlay network: the tree-based or the mesh-based (also called gossip-based). The first one organizes the peers in a tree structure, in which every peer receives the stream from its unique parent and forward the same signal to all its children; the second one organizes the peers in a mesh graph, with a quite casual number of parent/child relationships.

One of the main differences between tree- and gossip-based approaches is either the presence or not of a loop avoidance or detection procedure: only tree-based systems (Chuncky-spread, CrossFlux) avoid building loops, conversely mesh-based systems there is an attention to the best network configuration, based on partial knowledge of the overlay (CoolStreaming, Rainbow), on congestion control (Prime), on the number of friendship relations (Tribler), on the minimum RTT (GridMedia) or on an approximate solution of a particular optimization problem (LSONet). StreamComplete, although it is based on a mesh graph approach, provides a loop detection module plus a particular optimization procedure, in order to avoid bottlenecks and disadvantageous network configurations.

All the analyzed academic systems realize the join procedure (the phase in which a new peer connects itself to the overlay network) on the basis of a centralized node, that is, a login server or a bootstrapping node known a priori: this solution may cause a possible lack of scalability. StreamComplete, instead, provide a distributed join algorithm.

Together with procedures for overlay network's building and management, a P2P streaming system must decide which logical connections to use in order to transmit the video signal: this is the task of the so-called scheduling algorithm. In a tree-based system, the video stream must follow the entire previously built tree topology. In a mesh graph, conversely, the scheduling algorithm is more complex as of the greater freedom in choosing the signal routing.

Nowadays, there are not universal results to state that the mesh topology is better than the tree one, or vice versa. Indeed, the relative pros and cons are very clear. The mesh approach must balance the difficult trade-off between control traffic overhead versus delay in delivering packets. Instead, the tree-based systems have a continuity-versus-delay tradeoff. Namely, if in the tree a node's parent crashes or leaves the overlay network, then the same node will not receive packets until it find a new parent. If the node wishes to continuously play out packets to the application, then it must buffer enough packets to bridge the gap. So, the main difference between these two different approaches is that the mesh-based systems can be more failures and disconnections resilient but keeping low the traffic overhead it is not so easy, whereas in tree-based systems there is in general a more efficient videostream forwarding but with a big effort in order to guarantee the continuity of playback in case of churn. In particular, if a multi-tree network can spread one packet from the source to all other peers with O(n) forwarding, the gossip approach needs O(n log n) forwardings [4].

Finally, the robustness of P2P networks can be increased by using either data redundancy or path redundancy. The data are redundant because the stream is divided - but not partitioned - into several parts called stripes; redundant paths can be obtained using multiple neighbors nodes in sending the same stripe so that, in case of failures, remaining peers can continue to propagate the stream. The highest level of adaptiveness would serve all peers with

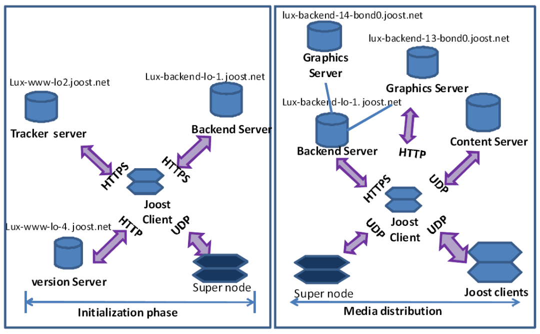


Fig. 2. Joost architecture

equal portion of bandwidth but, in a real context, peers offer heterogeneous upload speeds: this situation is the reason why the bottleneck are very frequent.

2.2 Commercial Systems

Being Joost and BabelGum two commercial and proprietary products, it is hard to learn the real technical solutions that they offer. However, [18] attempts to deduce the Joost architecture. As depicted in Figure 2, there are five different types of servers, Joost super nodes and Joost clients. There are also other servers in charge of various additional features (e.g., chat), but we skip them (and they are not shown in the figure) as our focus is on the streaming service.

The five types of servers participating in Joost are: version server, tracker server, backend server, channel graphics server and content server:

`lux-www-lo4.joost.net` **(version server)** is responsible for checking the current version of the software. When the Joost client (JC) is crashed, this server is also responsible for the error reports.

`lux-www-lo2.joost.net` **(tracker server)** during the initialization with a JC, sends the initial peer list that includes some of the super nodes and content servers. After that, the tracker server is not involved in any of the other stages (e.g., channel browsing, switching, etc.). Its only function is to keep track of members and help bootstrapping new peers.

`lux-backend-lo-1.joost.net` **(backend server)** is used by clients to perform channel list management (e.g., updating, downloading) and load balancing. Moreover, it periodically (every minute, 81 Kb traffic) communicates with the client. Indeed, the channel management is actually not performed by a single server, but by a server cluster. That is, the server `lux-backend-lo-1.joost.net` controls channel requests and keeps the load balance among the cluster servers, whereas other cluster servers are only responsible for a certain task (e.g., channel graphs downloading). E.g., there is a server, `lux-backend-13-bond0.joost.net` (`4.251.4.153`) from which the JC downloaded the channel graphs instead from the main server. Another task of the backend server is providing search services for channels or specific contents.

Graphics and content servers contain the delivered content. Joost deploy a serious number of such servers over the network.

Differently from other P2P applications (e.g., Skype) or overlay multicast solutions, super nodes are only used for controlling and helping new peers to find contributing peers. They are not responsible for relaying/forwarding media data to other peers. Peer management is split from the media distribution, which not only eases the management but also improves the efficiency of transmission. Differently, if a super node in Skype leaves ungracefully, all the other peers relying on it will be unavoidably affected.

The technical solutions of BabelGum are completely unreachable. We only say that BabelGum created a technology that compresses, decompresses and displays the video based on the standard H.264. The used ports are the 1111 or alternatively the 80 and 443.

2.3 Overlay Networks

In case of a tree-based topology, like in Chunkyspread and CrossFlux, a good practice is to organize a multi-tree structure for a better load balancing and robustness.

In ChunkySpread, the source (referred to as the true source) transmits the multicast stream to distinct slices. Each slice is transmitted over a multicast tree: it is very interesting that these trees are not necessarily node-disjoint.

Crossflux builds a multi tree overlay network, in which every node has one parent plus another *ghost* parent (or backup parent) that is able to guarantee the tree structure in case of first parent's departure. Each stream is divided - but not partitioned - into parts called stripes. The separation of the original file into stripes is performed by the source, which can optionally generate additional stripes for error correction. For each stripe one distribution-tree is created: this multi-tree approach increase the network reliability and allows to implement efficient recovery policies.

In case of mesh-based topology, it is possible to design very different solutions.

CoolStreaming is data-driven: every node periodically exchanges data availability information with a set of partners, and retrieves unavailable data from one or more suppliers. In the data-driven approach, a node always forward data to others that are expecting the data, with no prescribed roles like father/child, internal/external, upstreaming/downstreaming, etc.

In Rainbow the overlay network management is based on information about the geographical location of peer. Of course, the scenario of using Rainbow is the entire globe: if used only on relatively restricted geographical areas, the potential of the system would not be exploited to the maximum. An interesting feature of Rainbow is the two-level mesh-based overlay network. The lower layer consists of many clusters with a root node for each cluster. The cluster root nodes form the upper layer, the backbone network. The backbone network and each cluster are constructed as unstructured overlays with a fixed minimum node degree. In addition, random connections between members and clusters serve as shortcuts to reduce network delay and bandwidth consumption.

In Prime, participating peers form a randomly connected and directed mesh. All connections in the overlay are congestion controlled and are always initiated by the corresponding child peer. Each peer tries to maintain a sufficient number of parents that can collectively fill its incoming access link bandwidth.

LSONet manages one overlay network per sub-stream and it is based on principles very similar to CoolStreaming and Rainbow: the data-driven and the locality awareness.

The basic task of the overlay manager component on each GridMedia node is to find appropriate neighbors for each node by gossiping.

Finally, Tribler is based on the well known BitTorrent, plus an extension called cooperative download. An interesting add-on is called Tribler Streaming (TS), and it has been developed for the live video P2P streaming but, at the moment, its technical solutions are not reachable.

From the review, it stems that the overlay networks can be managed in several ways. The only universally accepted core principles are the locality awareness and the multi-path routing, wich represent the common features of all the analyzed systems.

2.4 Joining the System

In Chunkyspread, a new peer contacts a known-a-priori rendezvous node, which must know at least one member of the multicast group. The new node is then inserted within the group. In CrossFlux, a new peer contacts the known-a-priori source. The request is forwarded toward the child with the best state of health, so iteratively until finally the new node is added as a leaf.

In the mesh-based systems, there is a higher level of freedom in join-request forwarding and there can be a trade off between random and specific procedures. In CoolStreaming, a new peer contacts the known-a-priori source which forwards the request to a randomly chosen child. Rainbow is very different, as of the use of a specific login server that responds with a list of clusters' roots. The new peer must choose one of these and than can join the relative cluster. Prime, instead, creates more than one connection since the beginning: a new peer contacts a known-a-priori bootstrapping-node, which returns a random subset of nodes which start to directly serve the new peer. LSONet tries to optimize the topology since the join-request: a new peer contacts the nearest bootstrapping node which forwards the request to other peers. These peers either forward the request or accept it with probably *p=1/(1+sizeOf_partialView)*, where *partialView* is the overlay network subset known by the peer. Afterwards, the new node sends data requests to its suppliers basing on the approximate solution of an optimisation problem called OTP. GridMedia bases the joining phase on a well-known Rendezvous Point (RP), which is deployed to assist the construction of the overlay. During the startup, a participating node firstly contacts the RP to get a list of the nodes already in the overlay (called candidates list). The new participant will select several nodes from the candidates list as its initial neighbors. It first measures the Round-Trip Time (RTT) to each candidate and then chooses some nodes with the minimum RTT as one part of its initial neighbors. Finally, a Tribler peer, immediately after the first installation, automatically contacts, only once, one of a set of pre-known super-peers, in order to obtain an initial list of other peers in the system. So, it can start participating in the epidemic information dissemination. Then, according to the BitTorrent protocol, Tribler uses Web servers and tracker servers in order to find available data with the relative peers.

StreamComplete provides a completely distributed joining procedure processable by any peer without prefixed roles, increasing in this way the robustness and the scalability of the system.

2.5 Run-time Management

In case of peer disconnections or failures, the overlay network topology must be managed in order to guarantee the service without slowing down the performances. A continuous topology monitoring and management is needed in order to avoid or to solve the bottleneck into the overlay network: this is true for both tree- and mesh- based systems.

ChunkySpread uses a parent-switch operation in order to balance the workload and the latency values in the packet delivery. These operations are invoked by children and must be accepted by involved parents to avoid contemporaries switches.

Crossflux implements the so-called HeapTop algorithm to perform a father-child exchange when the father's bandwidth is significantly lower.

Also in this case, the mesh based systems are very heterogeneous. CoolStreaming does not provide a real optimisation procedure, and the management of the overlay network is based on three key modules: *(1)* membership manager, which helps the peers to maintain a partial view of other overlay nodes; *(2)* partnership manager, which establishes and maintains the partnerships with other nodes; *(3)* scheduler, which schedules the transmission of video data. The partnerships are established with a data centric procedure that is periodically re-invoked. In Rainbow the root-election procedure is started as often as possible in order to find the best network configuration. Prime bases the network management mainly on the congestion control: the links are added or removed on the basis of the bandwidth exploitation. LSONet, also in this case, is based on the OTP approximate solution, that is recalculated in the event of a change in data availability or in network configuration. GridMedia manages the neighborship of each node in a traditional distributed way after login process. And all the nodes will self-organize into an unstructured mesh. Each node has a member table initially obtained from RP, in which each item represents an active node that is currently in the overlay. The information of member tables is exchanged among neighbors periodically. Each node updates its member table in accordance with the member tables sent by its neighbors. Note that each item in the table has a field called life-time. It denotes the elapsed time since the latest message was received from this member. When the life-time of a node exceeds the predefined threshold, it will be removed from member table. Finally, Tribler is social-based, that is, the entire application is based on the following principle: the similarity in content tastes can be the founding principle for the construction of on-line communities with altruistic behavior. The Social Networking module in Tribler is responsible for storing and providing information regarding social groups (the group members, their recently used IP numbers, etc.). The context information that needs to be saved in order to improve the performance consists of information on social relations, altruism levels, peers uptimes, taste similarities, etc. In Tribler, every piece of context information received by a peer that is relevant to it, based on its interests and tastes, is stored locally in its so-called megacaches, and this information is exchanged within social groups using an epidemic protocol called Buddycast.

The state of art in run-time management shows that a pure probabilistic approach is not acceptable. All the P2P streaming systems provide some optimization policies, which can be classified into three families: event driven, periodic or the combination of the above.

Technology	**Overlay network**	**Joining**	**Run-time Management**	**Scheduling**	**Loop avoidance or detection**	**Core Principles**
CHUNKY SPREAD	Multi-tree structure not necessarily node-disjoint	A rendezvous node adds the new peer to one multicast group (tree)	Parent-switch operations in order to balance the workload.	The same rules of joining and run-time management	Avoidance	Node-degree management to carry the workload closer to the actual target value
CROSS FLUX	n couples of trees, one per stripe	The new peer contacts the root and finally it is added as a leaf	HeapTop algorithm to perform a father-child exchange	The same rules of joining and run-time management	Avoidance	Peer state of health and bandwidth availability monitoring
COOL STREAMING	Almost random mesh graph	The known-a-priori source forwards the request to a randomly chosen child	Data centric	The suppliers with greatest available bandwidth are preferred	None	Data-driven network and gossiping for membership management
RAINBOW	Network of clusters	A login server responds with a list of cluster roots.	Reorganisation of the clusters through the root election procedure	Pull-based scheduling mechanism to notify new available data	None	Locality-aware and pull-based data requesting
PRIME	Randomly connected and directed mesh	A bootstrapping-node returns a random subset of nodes	The links are added or removed on the basis of the bandwidth exploitation.	Pull-based dissemination of the video segments over the pre-built overlay	None	Reduce content- and bandwidth-bottlenecks
LSONet	One mesh graph per per sub-stream	Based on a bootstrapping node and on the approximate solution of an optimisation problem	Based on an approximate solution of an optimisation problem.	Based on an approximate solution of an optimal scheduling problem.	None	Solving an optimization problem in heuristic way, maximizing the bandwidth use
TRIBLER	Bit-Torrent based	Some super-peers provide an initial list of the others nodes	Based on the similarity in content tastes and the altruistic behavior's level.	With the cooperative download, a user invokes the help of his friends to speed up downloads.	None	The Bit-Torrent extension, called cooperative download

Technology	**Overlay network**	**Joining**	**Run-time Management**	**Scheduling**	**Loop avoidance or detection**	**Core Principles**
GRIDMEDIA	Almost random mesh graph	Some Rendezvous Points assist the construction of the overlay	Gossip based mechanism to exchange data, based on peers availability	Push-pull mechanism over the pre-built overlay	None	Push-pull scheduling procedure plus a periodic check of the neighborship.
STREAM COMPLETE	One workload-driven-mesh-graph per stream	Joining procedure processable by any peers without prefixed roles	Periodic data dissemination of availability and workload of the involved peers	Through a pull mechanism, the suppliers with greater state of health are preferred	Detection	Local optimization and auto-adaptability of the overlay topology

Table 1. Comparison of academic systems

2.6 Scheduling Algorithm

The scheduling algorithm manages the incoming and outgoing streams between peers, intervening on the network topology. In ChunkySpread and Crossflux, being tree-based, the algorithm coincides with the joining phase and with the run-time management of the network.

Instead, into a mesh-based system, each node knows a high number of others peers and the scheduling procedure can be implemented in different ways. In CoolStreaming, it is an heuristic algorithm based on the number of potential suppliers for each segment. Starting from segments with fewer suppliers, the peer chooses the supplier with the greatest available bandwidth. Rainbow provides a pull-based scheduling where each peer sends a notify message to its partners to declare the new available data. In Prime we have a two steps procedure: first, there is a diffusion phase in which segments are disseminated in a pull-based manner. In the second swarming phase, the data are sent from the peripheral peers toward the peers closer to the source. LSONet is based on an optimisation problem : given the network configuration and data and bandwidth capabilities, it is possible to define a problem of optimal scheduling of video streaming. The scheduling algorithm consists of the approximate solution of this problem, which is recalculated in the event of a change in the network configuration. In GridMedia, the local clock of each peer is synchronized with the Rendezvous Point. Furthermore, the system uses a push-pull mechanism: each node uses the pull method as a startup, and after that each node will relay a packet to its neighbors as soon as the packet arrives without explicit requests from the neighbors. Each node works under pure pull mode in the first time interval when just joining. After that, based on the traffic from each neighbor, the node will subscribe the pushing packets from its neighbors accordingly at the end of each time interval. Finally, Tribler provides the so-called cooperative download: a user invokes the help of his friends to speed up downloads. The peers that participate in a collaborative download take one of two roles: they are either collectors or helpers. A collector is the peer that is interested in obtaining a complete copy of a particular file, and a helper is a peer that is recruited by a collector to assist in downloading that file. Both the collector and the helpers start downloading the file using the classical BitTorrent Tit-for-Tat and the collaborative download extensions. After downloading a file chunk, the helper sends it to the collector without requesting anything in return.

Table 1 reports a synoptic view on the analyzed systems.

3 System Architecture

A StreamComplete peer is composed by three different logical layers (see Figure 3): NAO, DO & Controller, and View.

3.1 NAO Layer

This layer is the TCP gateway to the overlay network; it performs the mapping between Java objects and TCP packets (and vice versa) and it delivers the incoming messages to the right StreamComplete procedure. This gateway is used only for the control traffic, i.e., for the communications between the distributed algorithms.

In particular (see Figure 4(a)), `ListenerNAO` is a multithread TCP Server and starts in the boot phase; for each new connection, this TCP server starts a new thread using

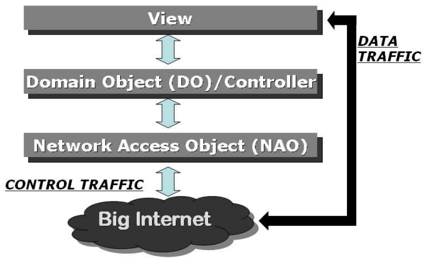
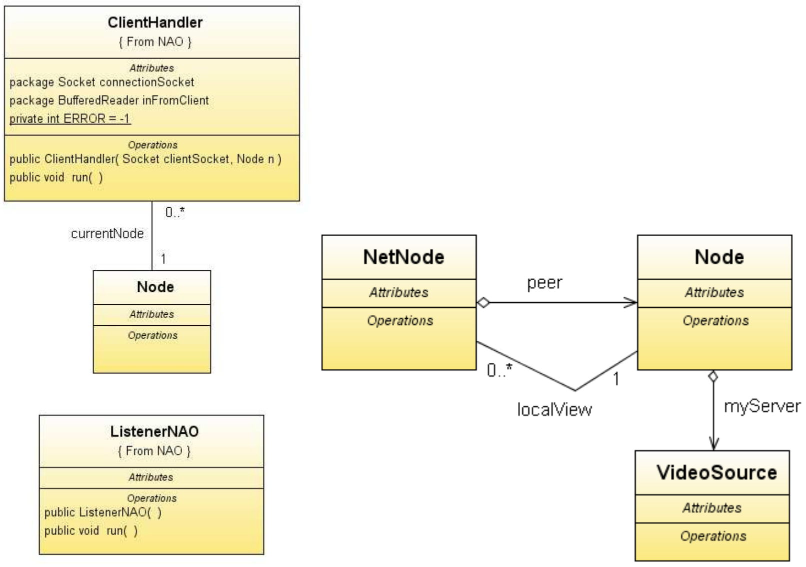


Fig. 3. StreamComplete architecture



(a) TCP multithread server (b) DO classes

Fig. 4. Details of the code objects

`ClientHandler`, which processes any new request received from other peers. Furthermore, it exists one NAO class for each procedure, which provides static methods for sending and receiving particular messages through the gateway. In particular, `ClientHandler` recognizes the type of received message and forward it to the appropriate NAO class. All the classes maintain the reference to the `Node` *me*, created in the boot phase, which represents the abstraction of the current peer.

3.2 DO & Controller Layer

It contains all the application logic; the algorithm instances manage the connections and the knowledge of the others peers. Part of the results of the computations are delivered to the View layer. The Domain Object (DO) package consists of three Java classes (see Figure 4(b)):

- `VideoSource` (also called `Server`) is the abstraction of the video source, i.e., the peer that generates the original video stream. The instance variables are the IP address and the port number, plus the identifier of the transmitted video;

- `Node` is the core domain object; it is the abstraction of the peer of the overlay network. It maintains a reference to a `VideoSource` object, to its own *local view* (its partial knowledge of overlay network) and to the objects of the View package. The current `Node` object interacts with the View package to manage the incoming and outgoing video streams which must be consistent with the state of the *local view.*

- `NetNode`: the local view of current node is a uniform set of `NetNode` objects; in each of these objects there is the reference to a `Node` object, plus two flags to know if that node is a supplier or a child or neither and the identifier of the possible video stream (from/to that node).

The Controller package is composed by two different types of classes:

- `*-Mex` classes, wich implement the abstractions of the TCP control packets; each procedure uses one or more types of messages to complete its own goal; this Java messages can have a timestamp, or a distinction between current sender and original sender (if the message is forwarded). Each Java message transports the state of health of the sender with some other information (node identifier, hop number, etc.). The core information contained in these messages is the `code` - that we can define like the header at the application level - that represents the procedure and the type of acknowledgement. Almost all of the messages are made only from header and, for this reason, the control traffc overhead in StreamComplete is very low.

- `Procedure` classes, wich provides methods to start the procedure or to reply to an external request: all the procedures are distributed and require interaction between two or more peers (see further in the following).

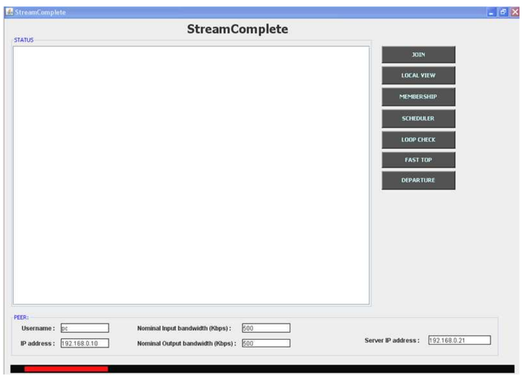
3.3 View Layer

It manages the incoming and outgoing video streams; it creates the right forwarding an the basis of the information received from the domain objects. While the control traffic is over the TCP, the data traffic is over UDP. All classes are based on the Java Media Framework (JMF) that enables audio, video and other time-based media to be added to applications and applets built with Java technology.

3.4 GUI

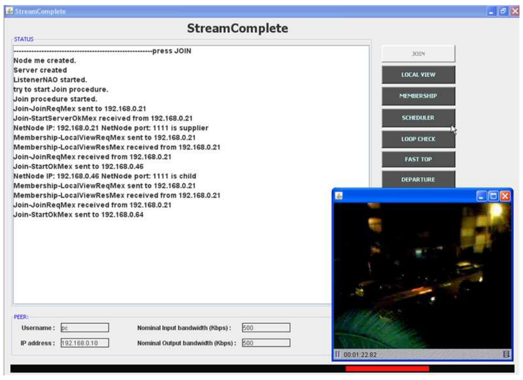
The user interface of StreamComplete has been designed in order to manage the tests (therefore it has not an appealing look&feel as commercial products). As the reader can see in Figure 5(a), there are three main areas: the first one, in the top left corner, prints the status messages; down, the second one shows the main information of the actual peer; finally, in the top right corner, there is a little control panel composed by seven buttons, each of them dedicated to invoke one StreamComplete's algorithm.

Once the user invokes the Join procedure, a new JMF window - with the received video stream - is placed in the bottom right corner, as shown in Figure 5(b).



Username : pc
IP address : 192.168.0.10
Nominal Input bandwidth (Kbps) : 500
Nominal Output bandwidth (Kbps) : 500
Server IP address : 192.168.0.21

(a) GUI



```
--------------------------------------------------------press JOIN
Node me created.
Server created
ListenerNAO started.
try to start Join procedure.
Join procedure started.
Join-JoinReqMex sent to 192.168.0.21
Join-StartServerOkMex received from 192.168.0.21
NetNode IP: 192.168.0.21 NetNode port: 1111 is supplier
Membership-LocalViewReqMex sent to 192.168.0.21
Membership-LocalViewResMex received from 192.168.0.21
Join-JoinReqMex received from 192.168.0.21
Join-StartOkMex sent to 192.168.0.46
NetNode IP: 192.168.0.46 NetNode port: 1111 is child
Membership-LocalViewReqMex sent to 192.168.0.21
Membership-LocalViewResMex received from 192.168.0.21
Join-JoinReqMex received from 192.168.0.21
Join-StartOkMex sent to 192.168.0.64
```

Username : pc
IP address : 192.168.0.10
Nominal Input bandwidth (Kbps) : 500
Nominal Output bandwidth (Kbps) : 500

(b) Incoming video stream

Fig. 5. StreamComplete Interface

4 STREAMCOMPLETE Algorithms

4.1 General Principles

As we discussed in Section 2, a possible approach to P2P streaming is to organize participating peers into multiple, diverse tree-shaped overlays where each specific sub-stream of the live content is pushed through a particular tree from the source to all interested peers. This approach has the following potential limitations: *(i)* in the presence of churn, maintaining multiple diverse trees could be very challenging; *(ii)* the rate of content delivery to each peer through individual trees is limited by the minimum throughput among the upstream connections; *(iii)* the outgoing bandwidth of those peers that do not have a sufficient number of children peers or an adequate amount of new content can not be effectively utilized. This in turn limits the scalability of the tree-based approaches.

The proposed architecture is based on a mesh overlay network and on several optimization procedures. In the same way as in a tree based network nodes less performing are leaves, the system centralizes the position of stronger peers: this is the core principle and it is implemented in an innovative algorithm called Fast-Top.

StreamComplete respects the small world property [b], but combines some aspects of this paradigm with the principle of locality awareness. Two main attractive properties of a small world network are: *(1)* low average hop distance between any two randomly chosen nodes, and *(2)* high clustering coefficient of nodes. Having a low average hop distance implies a low latency for object lookup, while having a high clustering coefficient implies the underlying network can effectively provide object lookup even under heavy demands (e.g., in a flash crowd scenario).

A way to construct a small world network is the following: *(1)* each node in the network is connected to some neighboring nodes, and *(2)* each node keeps a small number of links to some randomly chosen "distant" nodes. In a live video streaming scenario, is very important that every peer has many suppliers and is very important to avoid linear chains of nodes: all

[b] A small world network can be viewed as a connected graph in which two randomly chosen nodes are connected by just about six degrees of separation. In other words, the average shortest distance between two randomly chosen nodes is approximately six hops. This property implies that one can locate information stored at any random node of a small world network by only a small number of link traversals.

this is realized in StreamComplete through the Join and the Scheduling procedure and for this reason it is very close to the first principle of the small world paradigm. But in our scenario, the links to some randomly chosen distant nodes have to be avoided: this is due to the principle of locality awareness applied to the live video streaming. If we do not consider the network locality in the construction of the overlay, nearby hosts in the overlay network may actually be far away in the underlying network. This may waste too much network resources and, therefore, degrade performance significantly. For this reason StreamComplete allows nodes to increase their knowledge of the overlay network only in a "blemish oil" way: a peer knows its own neighbors, then the neighbors of the neighbors and so on. This is implemented in the Membership procedure. All this is necessary because the StreamComplete peers must organize themselves against the churn rate.

The probabilistic approach for network management is replaced in StreamComplete with a careful balancing of the workload across the overlay network. The key parameter used by StreamComplete is the *state of health*, which is calculated by every node and is disseminated within each communication. The state of health's value is based on: used incoming bandwidth (X), used outgoing bandwidth (W), nominal bandwidth value (Y), number of suppliers (Z) and number of hop from the video source (T). Each of this parameters contributes to the calculation of the state of health through a specific weight. The exact formula used is as follows (let H be the *health* function):

$$H(node) = \alpha \cdot X - \beta \cdot W + \chi \cdot Y + \delta \cdot Z - \epsilon \cdot T$$

In StreamComplete, the peers with a better health must serve a higher number of nodes and, to this end, they must position themselves in the inner part of the network. On the other hand, all the peers with a lower level of health must position themselves in the peripheral areas of the overlay network: this behavior resolves the problem of bottlenecks that is very likely in the P2P networks. Thanks to this particular network management, you can consider StreamComplete like a signal amplificator.

In the following of this section we analyze the four distributed algorithms designed in order to build and manage the overlay network.

4.2 Joining the System

The distributed algorithm designed for joining the system (simply called Join procedure) is able to connect the new node - in a completely distributed fashion - to more than one peer: to connect the new peer with only another one would be too risky in presence of churns.

If the new node starts the joining with the video source, this procedure creates exactly two connections. Furthermore, in any case, after the join to the system, the peers must start the membership and the scheduling procedures in order to increase their *local view.* There are three major steps in this procedure:

1. The new node X sends the request to one or more peers; *(1-bis)* on receiving the request, a peer calculates its own available outgoing bandwidth: if it is too low, it selects the best-health supplier with highest hop number and forwards the request to it, maintaining the information on the original sender;

2. Otherwise, it becomes directly itself one of the suppliers of X responding with a `StartOkMex`.

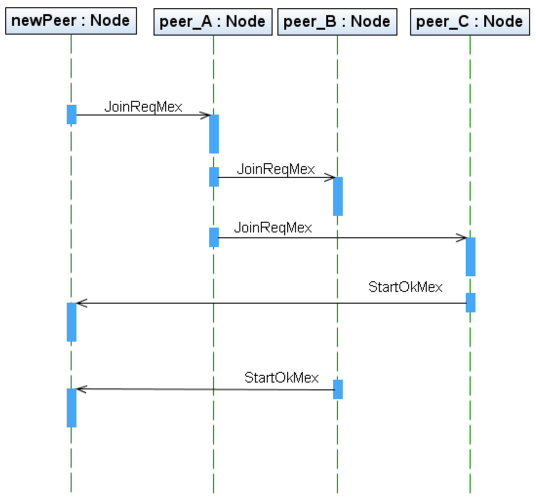


Fig. 6. Message pattern of the Join

This particular forwarding ensures that the request does not trace the overlay network too quickly back to the center, namely to the video source. So, StreamComplete is designed to be completely decentralized.

Being the Join parametric w.r.t. the first peer request, the new peer could directly contact the video source. In this case, the Join procedure tries to realize two connections: one from the video source itself and a second from a peer with high *state of health*. If the video source has not available outgoing bandwidth, the procedure realizes two connections with two generic peers with high *state of health* and high hop number. The innovation introduced in the Join consists in connecting the new node to the peripheral areas of the mesh overlay network. Furthermore, the procedure tries to avoid the formation of linear chains in order to keep low delays.

Figure 7 depicts a message pattern summarizing the major steps of the procedure, whose pseudo-code is presented in Algorithm 1.

4.3 Membership

During its own lifetime, a peer increases and upgrades its partial knowledge of the overlay network thanks to the Membership procedure, which exchanges information between couples of peers. So, each peer maintains a local view of the overlay network. The local view's cardinality must be limited in order to guarantee the scalability of the system. In each peer, the knowledge of the overlay network starts with the neighbor peers and increases to the farthest, like an "oil spot".

In only two step, the peers exchange their local views. This procedure can be started by any peer at any time.

1. The peer that starts the procedure sends a `LocalViewReqMex` message to all the other known peers. This particular message contains all the values of the sender's local view.

```
Input: currentNode: Node; videoSource: Node;

Upon Event [INIT];
JoinReqMex jm;
jm.sendTo(anotherNode);

Upon Event [JoinReqMex jrm DELIVER | originalSender, sender];
if currentNode.isTheVideoSource() then
    if available_input_bandwidth < x% of nominal_input_bandwidth then
        Node first = first best known Node with high hop number;
        Node second = second best known Node with high hop number;
        jrm.forwardTo(first);
        jrm.forwardTo(second);
    else
        currentNode.children ∪= {originalSender} ;
        StartServerOkMex ssom;
        ssom.sendTo(originalSender);
        Node first = first best known Node with high hop number;
        jrm.forwardTo(first);
    end
else
    if available_input_bandwidth < x% of nominal_input_bandwidth then
        Node first = first best Supplier with high hop number;
        jrm.forwardTo(first);
    else
        currentNode.children ∪= {originalSender} ;
        StartOkMex som;
        som.sendTo(originalSender);
    end
end

Upon Event [StartServerOkMex mex DELIVER | videoSource];
currentNode.suppliers ∪= {videoSource} ;
currentNode.hopNumber = 1;

Upon Event [StartServerNotOkMex mex DELIVER | videoSource];
STOP;

Upon Event [StartOkMex mex DELIVER | sender];
currentNode.suppliers ∪= {sender} ;
currentNode.hopNumber = $min\{x | x = currentNode.suppliers[i].hopNumber \, \forall i\} + 1$ ;
```

Algorithm 1: Join

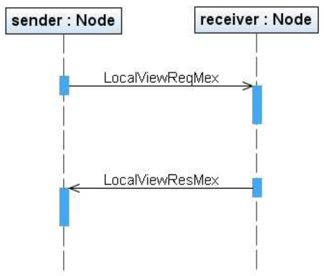


Fig. 7. Message pattern of the Membership

2. The peer that receives a `LocalViewReqMex` message updates its local view, and finally replies with a `LocalViewResMex` message in which there are only information unknown to the other peer.

This procedure has important features: *(i)* it has a minimum number of steps, only two; *(iii)* the response message does not contain the same information of the initial request. These properties minimize the traffic overhead introduced by the procedure.

```
Input: $currentNode$: Node; $receiver$: Node;
Upon Event [INIT];
LocalViewReqMex $lvrm$;
$lvrm$.sendTo($receiver$);

Upon Event [LocalViewReqMex $mex$ DELIVER | senderLocalView, sender];
$currentNode$.localView $\cup = \{senderLocalView\}$ ;
Node[ ] $response$ = $currentNode$.localView - $senderLocalView$;
$response$.sendTo($sender$);

Upon Event [LocalViewResMex $mex$ DELIVER | senderLocalView, sender];
$currentNode$.localView $\cup = \{senderLocalView\}$ ;
```

Algorithm 2: Membership

Figure 7 depicts a message pattern summarizing the major steps of the procedure, whose pseudo-code is presented in Algorithm 2.

4.4 Scheduling

In a P2P live streaming scenario, scheduling means to change the network topology to increase the local performance. In StreamComplete, a peer can find new potential suppliers by increasing its own incoming degree: higher used-incoming-bandwidth means higher state of health. Thanks to the local view, each peer can create new incoming or outgoing video streams: this video-stream-routing procedure (called Scheduling) also aims to balance the workload on the basis of the suppliers' state of health. The goal of Scheduling procedure is to fill the incoming bandwidth with the best available suppliers: this behavior can allow to

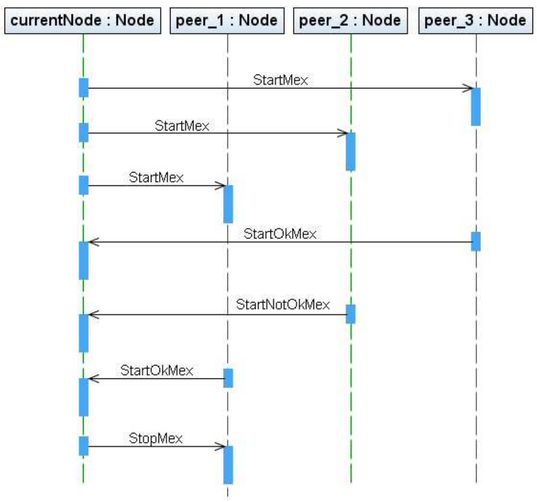


Fig. 8. A possible message pattern for the Scheduling

survive even in the case of major phenomena of churn (the continuous addition/removal of users to the system). Below, the three steps of the procedure:

1. With a new thread, a peer starts the scheduling procedure and, in this first step, orders all its known peers by decreasing the state of health and sends to them a `StartMex` message.
2. On receiving a `StartMex` message, a peer checks its available outgoing bandwidth and, if it is not too low, accepts the request sending a `StartOkMex` message; otherwise the peer sends a `StartNotOkMex` message.
3. Note that the number of new suppliers that the peer can accept is limited, so, if it receives a positive response, it must check the available input bandwidth: if this is too low, the peer replies with a `StopMex` message to interrupt the logical link; otherwise it begins to receive the video stream. If the peer receives a negative response, it updates the local-view data only.

Figure 8 depicts a message pattern summarizing the major steps of the procedure, whose pseudo-code is presented in Algorithm 3.

4.5 FastTop

The proposed architecture is based on a mesh overlay network and on several optimization procedures. As well as in a tree-based network you prefer less performing nodes as leaves, StreamComplete centralizes the position of stronger peers: this is the core principle and it is implemented in FastTop. It derives from this particular union and gives an “intelligent” behavior to the network: to increase the general performance of the overlay network, it is important to avoid the bandwidth bottleneck. Suppose that the health of a peer X is much

```
Input: $currentNode$: Node;
Upon Event [INIT];
foreach $Node\ n \in currentNode.localView$ do
    StartMex $sm$;
    $sm$.sendTo(n);
end

Upon Event [StartMex $mex$ DELIVER | sender];
if $sender \in currentNode.suppliers \vee sender \in currentNode.children$ then STOP;
else if $available\_output\_bandwidth >= x\%\ of\ nominal\_output\_bandwidth$ then
    $currentNode$.children $\cup = \{sender\}$ ;
    StartOkMex $som$;
    $som$.sendTo($sender$);
else
    StartNotOkMex $snom$;
    $snom$.sendTo(sender);
end

Upon Event [StartOkMex $mex$ DELIVER | sender];
if $available\_input\_bandwidth >= x\%\ of\ nominal\_input\_bandwidth$ then
    $currentNode$.suppliers $\cup = \{sender\}$ ;
else
    StopMex $stopm$;
    $stopm$.sendTo(sender);
end

Upon Event [StartNotOkMex $mex$ DELIVER | sender];
STOP;

Upon Event [StopMex $mex$ DELIVER | sender];
$currentNode$.children -= $\{sender\}$ ;
```

Algorithm 3: Scheduling

greater than its supplier Y: the network may have beneficial effects if we reduce the hops of X. To this end, the low-state-of-health peer must position itself in the peripheral areas of the network (almost like a leaf in a tree-based architecture) while all the peers with high state of health must trace the network back to the video server and back to the central areas (almost like the children of the tree root). In a mesh graph, swapping the position of two nodes is very difficult. You can think to a view of a graph ordered by increasing number of hops from a particular node, the video server: while in a tree there is only one father for each node (excluding the root), in a mesh graph the links are very complex and a swapping procedure must manage this complexity.

There are two main ways to implement this procedure.

A. We could reverse the two nodes in a transactional way. But in this case, swapping the roles of X and Y is very difficult (see Figure 9(a)) because it means that we must exchange the suppliers and the children of these two peers. This type of procedure requires a single transaction and this is not recommended in a very dynamic scenario with high churn. So this first solution is not good and must be avoided.

B. There is another solution that does not need transactions and that allows to reduce the hop number of X (see Figure 9(b)). The main principle is that X must receive the stream from the suppliers of Y: even this operation decrease the hop number. At the same time, it is important that Y moves itself under X, that is, it is important that X become its only supplier. This procedure must be started by the original child only if the state of health of the supplier is extremely low.

StreamComplete uses this second solution through the following six steps procedure, whose pseudo-code is presented in Algorithm 4.

1. The first step consists in the searching of the lowest health-state supplier: if the value is lower than a particular percentage, the peer X can start the procedure by sending the request messages (to the peer Y, for example);

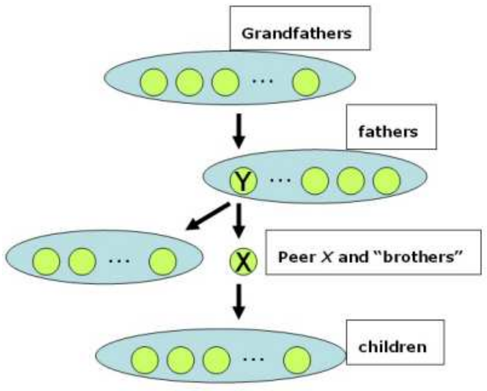
2. On receiving a request message, peer Y checks its current health state to verify the real gap with its child (the starter peer). So, peer Y replies with a positive or negative *ack* message;

3. In case of positive *ack*, the peer Y itself must stop all the video streams received from its own suppliers;

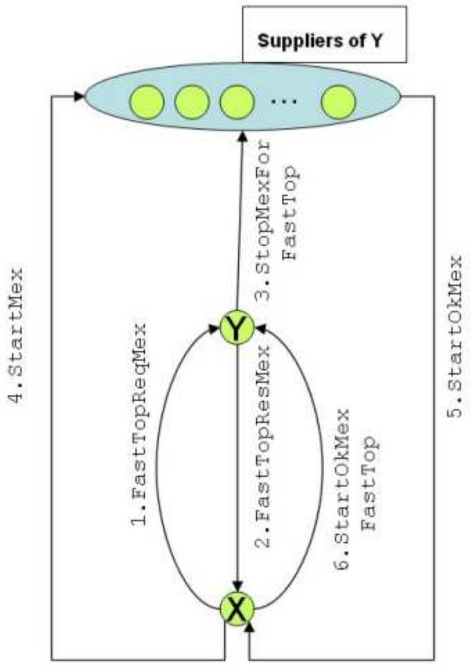
4. In case of negative *ack* message, the peer X only updates its own local view and stops the procedure; otherwise, the positive *ack* message contains the list of Y's suppliers: in this case, after updating the local view, the peer X sends a request messages to them in order to receive new video streams;

5. On receiving a positive *ack* message from the new suppliers, peer X can accept the new video streams;

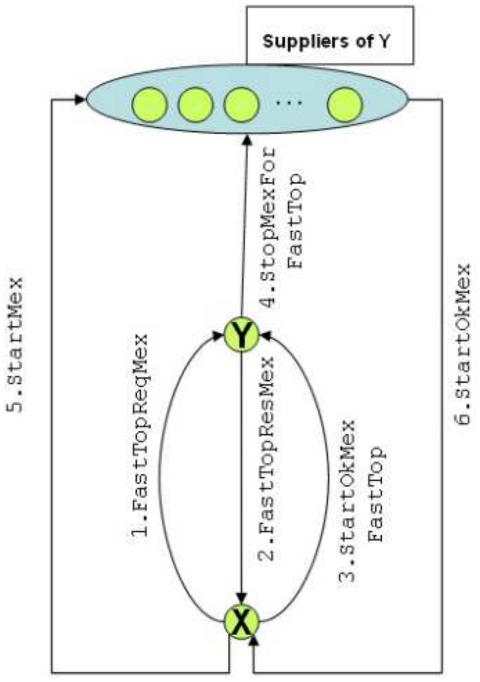
6. At same time, it must start a new video stream to Y which becomes, now, a child.



(a) How to swap the roles of X and Y ?



(b) The six steps



(c) The final solution

Fig. 9. Fast Top

Nevertheless, there is a problem in this solution: there is a time interval in which peer Y does not receive any video stream: in particular from step three to step six. So, we must re-organize the evolution of the procedure simply as follow (see Figure 9(c)):

1. `FastTopReqMex` from peer X to peer Y;
2. `FastTopResMex` (positive or negative) from peer Y to peer X;
3. `StartOkMexFastTop` from peer X to peer Y;
4. `StopMexByFastTop` from peer Y to all its suppliers;
5. `StartMex` from peer X to all its new suppliers;
6. `StartOkMex` from the suppliers to peer X.

4.6 LoopCheck

Generally, the mesh-based P2P video streaming systems are developed without a loop avoidance or a loop check mechanism. In particular, a loop avoidance procedure would be too disadvantageous from a computational point of view: before adding a new link in the overlay network, a peer should check the presence of a potential loop that involves the new link itself but this mechanism would slow down too much all core procedures (Join, Scheduling, Fast Top). In the "zapping" phase of a user, the system must have a *near real time interaction.* Our loop check mechanism does not slow the system because *(i)* it is invoked after the end of other procedures and *(ii)* it is correct but not complete [c]. In particular:

1. With a new thread, a peer sends to all its children a `LoopCheckMex` message which contains: the orignial sender, the current sender, the current receiver, the bound of the possible loops and the TTL. In this first message, TTL and bound have the same value. A good practice is to set the bound equal to the size of the local view.
2. The received `LoopCheckMex` messages are analyzed: if no loop is found, the current peer forwards the message to its children with the decreased TTL (until it is greater than zero). If the message's original sender is the current node itself, then there is a loop: in this case the peer cuts off the logical link to the last sender of the message (i.e., a supplier).

Figure 10 depicts a message pattern summarizing the major steps of the procedure, whose pseudo-code is presented in Algorithm 5.

[c] Correctness: a loop check algorithm LC is correct if, for each graph g, when $LC(g)$ = `true` then g actually contains a cycle. Completeness: a loop check algorithm LC is complete if, given a graph g, when g contains a cycle then $LC(g)$ = `true`. A correct and complete loop check algorithm, in case of n nodes graph, must check all the possible i-hops paths where $i = 2,\ldots,n$.

```
Input: $currentNode$: Node; $candidate$: Node; $gap$: int;
Upon Event [INIT];
FastTopReqMex $ftreqm$;
$ftreqm$.sendoTo($candidate$);

Upon Event [FastTopReqMex $mex$ DELIVER | child];
if $currentNode.health \geq gap\%$ of $child.health$ then
    FastTopResNotMex $ftnot$;
    $ftnot$.sendoTo($child$);
else
    FastTopResYesMex $ftyes$;
    $ftyes$.sendTo($child$);
    foreach $Node\ n \in currentNode.suppliers$ do
        StopMexByFastTop $ftstop$; $ftstop$.sendTo(n);
        $currentNode$.suppliers -= $\{n\}$;
    end
end

Upon Event [FastTopResNotMex $mex$ DELIVER | sender];
STOP;

Upon Event [FastTopResYesMex $mex$ DELIVER | sender];
foreach $Node\ n \in sender.suppliers$ do
    StartMex $sm$;
    $sm$.sendTo(n);
end

Upon Event [StopMexByFastTop $mex$ DELIVER | sender];
$currentNode$.children -= $\{sender\}$;

Upon Event [StartMex $mex$ DELIVER | sender];
$currentNode$.children $\cup = \{sender\}$;
StartOkMex $ftok$;
$ftok$.sendTo(sender);

Upon Event [StartOkMex $mex$ DELIVER | sender];
$currentNode$.suppliers $\cup = \{sender\}$;
StartOkMexFastTop $ftsok$;
$ftsok$.sendTo($candidate$);

Upon Event [StartOkMexFastTop $mex$ DELIVER | sender];
$currentNode$.suppliers $\cup = \{sender\}$;
```

Algorithm 4: FastTop

```
**Input**: $currentNode$: Node;
Upon Event [INIT];
foreach $Node\ n \in currentNode.children$ do
    LoopCheckMex $lcm$;
    $lcm$.sendTo($n$);
end
Upon Event [LoopCheckMex $mex$ DELIVER | sender];
if $sender \neq currentNode$ then
    if $mex.TTL > 0$ then
        $mex$.TTL -= 1;
        foreach $Node\ n \in currentNode.children$ do
            $mex$.forwardTo($n$);
        end
    else
        STOP;
    end
else
    $currentNode$.suppliers -= $\{sender\}$;
    STOP;
end
```

Algorithm 5: LoopCheck

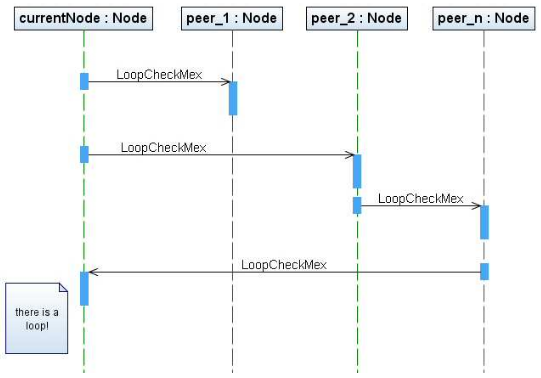


Fig. 10. A possible message pattern of the Loop Check

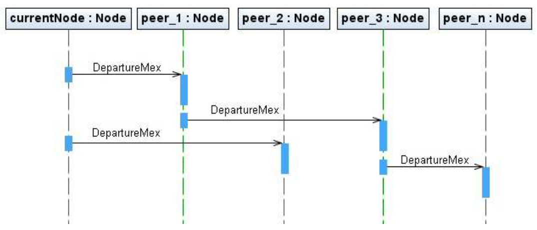


Fig. 11. A possible message pattern of the Departure

4.7 Departure

A peer that leaves the overlay must warn the others nodes: to this end, the Departure procedure achieves a controlled network flooding warning. Each message is forwarded a limited number of times, generally equal to the local view's cardinality. After a departure, the local overlay region re-organizes the network topology through the Scheduling and Membership procedures. In particular:

1. the departing peer sends a `DepartureMex` message to all the nodes of its local view, with the TTL set to bound value.
2. On receiving a `DepartureMex` message, a peer deletes the original sender from its own local view and then it forwards the message - with decreased TTL - to all known peers.

```
Input: $currentNode$: Node;
Upon Event [INIT];
foreach $Node\ n \in currentNode.localView$ do
    DepartureMex $dm$;
    $dm$.sendTo($n$);
end
Upon Event [DepartureMex $mex$ DELIVER | sender];
$currentNode$.localView -= $\{sender\}$;
if $mex.TTL > 0$ then
    $mex$.TTL -= 1;
    foreach $Node\ n \in currentNode.localView$ do
        $mex$.forwardTo(n);
    end
end
```

Algorithm 6: Departure

A possible message pattern is as shown in Figure 11, and the simple pseudo-code is in Algorithm 6.

5 Experimental Evaluation

We have extensively evaluated StreamComplete over the PlanetLab. As of the strong constraints of the PlanetLab testbed, the user interface is substituted by an appropriate module that simulates the bandwidth use. Tests ran over almost all the nodes of PlanetLab, with a number of different nodes that ranges from about 150 to 400 or over, with a maximum of about 800 (April, May and June, 2008). The StreamComplete prototype has been deployed through the *PlanetLab Experiment Manager* (PlMan) software of the University of Washington. Thanks to the PlMan, it is easy to send SSH commands to all the slice' nodes.

The video source peer was the `planetlab-1.dis.uniroma1.it` node. In each node, through a local MySQL database, we measured the performance of the algorithms, in terms of time and generated control traffic. Furthermore, we log the evolution of the local-views.

In order to reproduce realistic scenarios, with heterogeneous peers and variable churn rate and video frame rate, we conducted multiple executions of the system with different parameters. To this end, we fixed a set of *session parameters*, in particular:

1. **video frame rate**: the video frame rate ranges from about 30 Kbps to 100 kbps, depending on the quality of the video capture device;

2. **churn rate**: in the video streaming context, the rate of departing or joining of the peers depends on either failures and the "zapping" of the users themselves across the available videos. In particular, ever x minutes there is a range *[w,t]* of time in which peers can disconnect and re-connect themselves with probability p;

3. **max local-view cardinality**: each peer has not to know all the overlay network's nodes, for this reason StreamComplete limits the maximum local-view cardinality in a range from 10 to 100 peers.

4. **available bandwidth**: in our experiments, each peer can have a xDSL connection that range in the following set: 2 Mbps, 4 Mbps or 7 Mbps.

Furthermore, the *state of health* calculus is as follow (let H be the *health* function):

$$H(node) = 0.2 \cdot X - 0.35 \cdot W + 0.25 \cdot Y + 0.15 \cdot Z - 0.05 \cdot T$$

StreamComplete has been tested into five realistic scenarios, in order to simulate five possible applications of such a system. The first one is about the *professional entertainment* which assumes a high video frame rate and a quite high maximum local-view cardinality. Because of the high video quality, the 50% of the population has the fastest internet connection and only the 20% has the slowest. The churn rate is medium. So, in this first scenario we assumes the presence of a professional provider and of nodes with high performance.

Below, all the details:

- Video frame rate: 100 kbps;

- Churn rate: x=3, w=0, t=2, p=0.4;

- Max local-view cardinality: 60 peers;

- Nodes bandwidth population: 20% @ 2 Mbps, 30% @ 4 Mbps, 50% @ 7Mbps.

The second scenario simulates an *amatorial entertainment* context and for this reason it is quite similar to the first one, even if it presents lower video-quality and a consequent higher maximum local-view cardinality. The typical context is a P2P television with user generated contents. Below, all the details:

- Video frame rate: 60 kbps;
- Churn rate: x=3, w=0, t=2, p=0.4;
- Max local-view cardinality: 100 peers;
- Nodes bandwidth population: 20% @ 2 Mbps, 30% @ 4 Mbps, 50% @ 7Mbps.

The third scenario is about *private security*, for example the surveillance of a private office, in which only the deployment of few high-quality devices is requested. In this particular scenario the churn rate is very low, and only two kinds of nodes exist: devices with slow connection (PDAs, wireless devices, etc.) and high performance nodes (i.e., the nodes of the control center). Below, all the details:

- Video frame rate: 60 kbps;
- Churn rate: x=3, w=0, t=2, p=0.3;
- Max local-view cardinality: 20 peers;
- Nodes bandwidth population: 20% @ 2 Mbps, 0% @ 4 Mbps, 80% @ 7Mbps.

The fourth scenario is quite similar to the previous, since it simulates the public security, in particular the *environmental monitoring*: monitoring of streets, public places, mountains, volcanos, etc. In this case, the video quality could be lower than the previous but, at the same time, the maximum local view cardinality could be higher and the nodes bandwidth population more complex, because of legagy devices or slow wireless connections. Below, all the details:

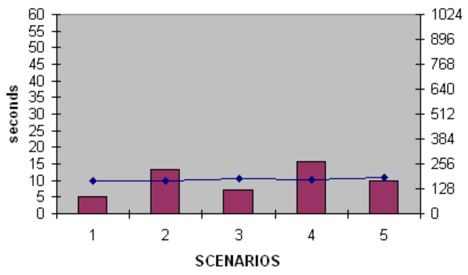
- Video frame rate: 50 kbps;
- Churn rate: x=3, w=0, t=2, p=0.3;
- Max local-view cardinality: 40 peers;
- Nodes bandwidth population: 50% @ 2 Mbps, 30% @ 4 Mbps, 20% @ 7Mbps.

Finally, the fifth scenario has been designed in order to put StreamComplete under stress and in order to study the ability of the system to react in case of a very dinamic scenario. For all this reasons, the fifth scenario is called *stress*. Below, all the details:

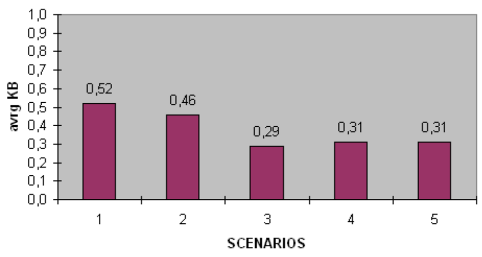
- Video frame rate: 40 kbps;
- Churn rate: x=2, w=0, t=1, p=0.7;
- Max local-view cardinality: 100 peers;
- Nodes bandwidth population: 60% @ 2 Mbps, 0% @ 4 Mbps, 40% @ 7Mbps.

SCENARIO	**Num. of runs / Num. of messages**
1 - Professional Entertainment	6,604 / 144,571
2 - Amatorial Entertainment	4,460 / 121,527
3 - Security	2,907 / 70,217
4 - Environmental Monitoring	4,215 / 142,777
5 - Stress	3,950 / 105,731

Table 2. Data collected during testings



(a) Performance



(b) Average control traffic

Fig. 12. Join procedure

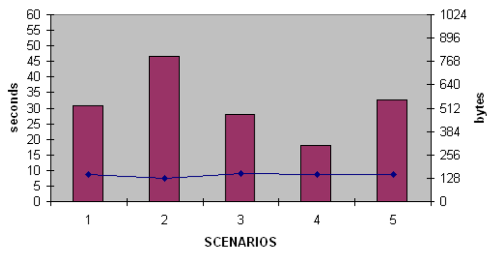
We totally collected data on 22.136 runs of the algorithms with 584.823 messages exchanged between the nodes.

The traffic and time data are shown into double charts, in which the bar chart represents the time performance (relative scale on the left) and the line represent the average size of control packets (relative scale on the right); everything is grouped by scenarios.

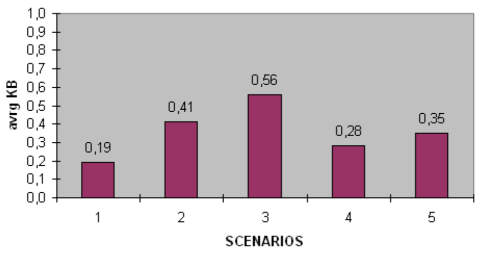
Join Procedure. Figure 12(a) demonstrates the ability of peers of joining the system and receiving the video stream in only few seconds. As shown in the figure, the scale of seconds ranges from 0 to 60, being the initialization period of DOnet [7] of about one minute. In the first scenario we registered an average time of 5.2 seconds and 15,5 in the fourth one: this range depends on the current workload of PlanetLab. Note that in the *stress* scenario we registered an average time of only 9.9 seconds. The average size of control packet is about 180 bytes, which is irrelevant in terms of total traffic.

Furthermore, we analyzed the control traffic of the *Join* procedure. Each peer, during the joining phase, generates a very low control traffic. This is a very important result that demonstrates the ability of StreamComplete in solving one of the main problems of the P2P networks: the control traffic's overhead. In particular, Figure 12(b) shows the average control traffic of a single run, for each scenario. Note that, in joining the overlay network, each peer generates, in average, at most 0.52 KB.

FastTop. Figure 13(a) shows the performance of FastTop. In the different scenarios we registered times that range from 18 to 46 seconds, including disconnections and the re-activations of all the involved video streaming. Note that FastTop is much more complex and distributed than all the others algorithms. Furthermore, this procedure is scalable w.r.t. the overlay workload: in the fifth scenario, which is the more stressful, we registered a result similar to the others (average of 32.795 seconds with a minimum of 1.694 seconds). So, StreamCom-

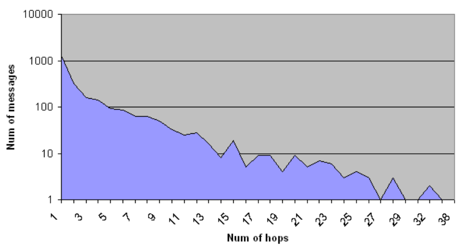


(a) Performance

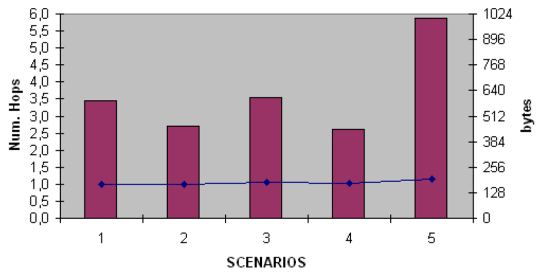


(b) Average control traffic

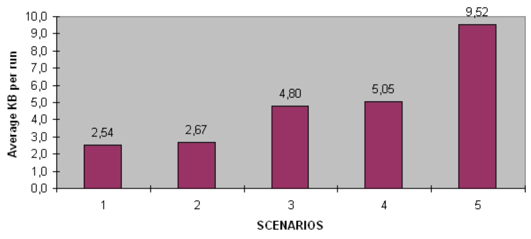
Fig. 13. Fast Top procedure



(a) Distribution of number of hops in the first scenario



(b) Loop Check performance, grouped by scenario



(c) Loop Check - average traffic (KB) per run

Fig. 14. Loop Check procedure

PLETE reacts very well in case of very dynamic cases with high churn rate. The average size of the control packet is about 150 bytes.

Another import result of our experiments is the very low average control traffic of FastTop. This complex procedure, in average, takes at most 0.56 KB per *run* (see Figure 13(b)). So, this procedure can be used very often without slowing down peers and, at the same time, increasing the overall performance of the overlay network.

Obviously, the FastTop procedure is invoked only in particular network configuration and, for this reason, the number of runs is quite low.

Loop Check. We analyze some important aspects: first, the average number of hops of the messages, that is, the average number of forwarding of each message; second, the generated traffic. The results are directly proportional to the configuration of the scenarios. For example, the *stress* one: in this case we registered the highest average hops number and this depends on the local view cardinality and on the high out-degree of nodes. In any case, the design

SCENARIO	**Average hops number**
1 - Professional Entertainment	3.44
2 - Amatorial Entertainment	2.69
3 - Security	3.55
4 - Environmental Monitoring	2.61
5 - Stress	5.87

Table 3. Loop Check Performance, grouped by scenario

of all the procedures avoids, in most cases, the formation of loops, since the joining to the system.

In average, the majority part of the Loop Check message is forwarded at most six times. For example, Figure 14(a) shows the details of the Loop Check procedure in the first scenario (professional entertainment): note that the y-axis is logarithmic and the 84,71% of all the messages are forwarded at most six times. Being a `LoopCheckMex` forwarded from a peer to each own child, the number of hops also represents the average length of the walks of the video stream. So, a low average hops number confirms that StreamComplete can organize the mesh graph in compliance with two main properties of the small world networks.

From the point of view of generated traffic, Figure 14(c) shows very low values in average KB per runs. Obviously, in this case the generated traffic is a little more than the other analyzed procedures, since the Loop Check makes a partial flow of the overlay network. In any case, comparing this values with a realistic amount of KB of the video traffic, Loop Check provides excellent result like the other algorithms.

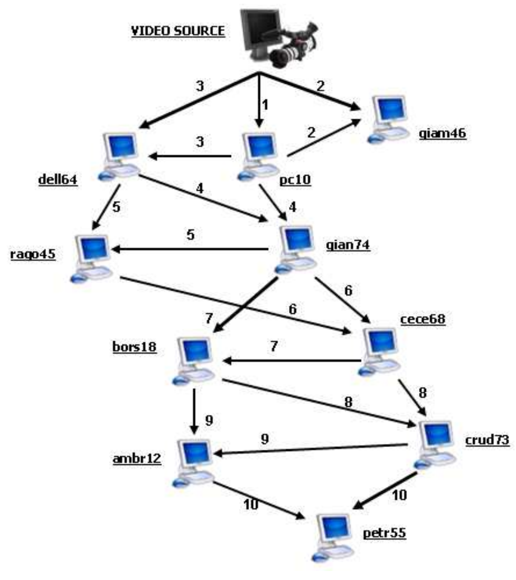
5.1 Video Demonstration

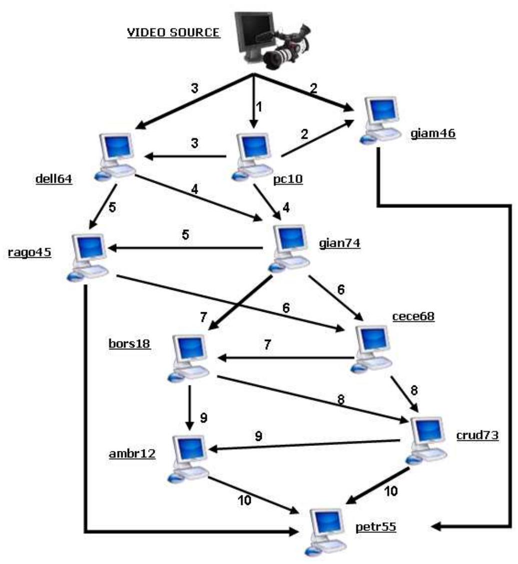
We have realized a video demonstration of the Join procedure, available at `www.dis.uniroma1.it/pub/mecella/projects/StreamComplete`. The video shows a running of StreamComplete, made with eleven peers and using a non-professional video-capture device. The evolution of the initial join phase is described through black arrows (see Figure 15(a)). The first peer that requests the service is *pc10*, followed by *giam46*, *dell64* and so on: the arrows' labels show the temporal evolution of the joining. Note that StreamComplete avoids linear chains since this phase and, at the same time, it makes a first workload balancing through the double initial connections.

In this demonstration (for reasons of visual clarity of the video), in addition to the join phase, only one execution of the Scheduling procedure is shown, and it is invoked by the last peer, called *petr55* (see Figure 15(b)). The peers *rago45* and *giam46* are very close to the video source and have a very low number of outgoing connection: finally, they are the two best state of health peers and the Scheduling procedure is able to detect them in order to fill up the *petr55*'s incoming bandwidth - creates a new video stream from the two best peers - from the free outgoing bandwidth's point of view.

6 Conclusions

Every P2P network is a computing power and transmission amplifier. With the increasing connectivity and bandwidth availability, it is possible to develop critical systems in a P2P





(a) An evolution of the Join Procedure (b) *Petr55* invokes the Scheduling procedure

Fig. 15. Demonstration

fashion, especially in case of multimedia applications. StreamComplete realizes local optimizations thanks to an autonomic behavior of the peers. The tests of the proposed system demonstrates the low time of connection, the ability of peers in re-organizing the network topology in a completely distributed manner and the very low traffic overhead.

References

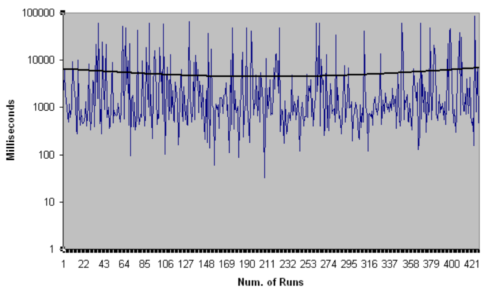
1. X. Ye, J. Zhang, J. Wang. *Architecture of HIKEC: An IMS-based Mobile P2P File Sharing Service.* Proc. ICCT 2006.
2. J.H. Kim, J.W. Song, T.H. Kim, J.H. Lee, K.J. Lee, J.S. Han, S.B. Yang. *Reliability Schemes for P2P Systems in Dynamic Mobile Environments.* Proc. CSA 2008.
3. G. Gehlen, F. Aijaz, Y. Zhu, B. Walke. *Mobile P2P Web Service Creation using SIP.* Mobile Information Systems, 3(3-4): 165-185, 2007.
4. S. Agarwal, S. Dube. *Gossip Based Streaming with Incentives for Peer Collaboration.* Proc. ISM 2006.
5. V. Venkararaman et al. *Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast.* Proc. ICNP 2006.
6. M. Schiely, P. Felber. *Tit-for-tat Revisited: Trading Bandwidth for Reliability in P2P Media Streaming.* Technical Report 2007.
7. X. Zhang, J. Liu, B. Li, P. Yum. *CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming.* Proc. INFOCOM 2005.
8. N. Magharei, R. Rejaie. *PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming.* Proc. INFOCOM 2007.
9. N. Magharei, G. Yang, R. Reza. *Issues in Offering Live P2P Streaming Service to Residential Users.* Proc. CCNC 2007.
10. H. Guo, K.T. Lo, C.T. Cheng. *Overlay Networks Construction for Multilayered Live Media Streaming.* Proc. ISM 2006.

11. Y. Chen, B. Deng, X. Li. *Rainbow: a Locality-aware Peer-to-Peer Overlay Multicast System.* Proc. Grid and Cooperative Computing Workshop, 2006.
12. Joost, web site: `http://www.joost.com/`
13. BabelGum, web site: `http://www.babelgum.com/`
14. J.A. Pouwelse, P.Garbacki, J.Wang, A.Bakker, J.Yang, A. Iosup, D.H.J.Epema, M.Reinders, M.R.vanSteen, H.J. Sips. *TRIBLER: a social-based peer-to-peer system.* 2007 John Wiley & Sons.
15. K.Y.K. Hui, J.C.S. Lui, D.Y. Yau. *Small World Overlay P2P Networks.* Proc. IWQOS 2004.
16. F. Covino, M. Mecella *STREAMCOMPLETE: an Architecture for Mesh-based Peer-to-Peer Live Video Streaming.* Proc. CCNC 2009.
17. B. Li, H. Yin *Peer-to-Peer Live Video Streaming on the Internet: Issues, Existing Approaches, and Challenges.* IEEE Communications Magazine, 2007
18. J. Lei, L. Shi, X. Fu *An Experimental Analysis of Joost Peer-to-Peer VoD Service.* Technical Report No. IFI-TB-2007-03, Institute of Computer Science, University of Goettingen, Goettingen, Germany, October 2007.
19. M. Zhang, L. Zhao, Y. Tang, J.G. Luo, S.Q. Yang *Large-Scale Live Media Streaming over Peer-to-Peer Networks through Global Internet.* Proc. P2PMMS 2005.
20. M. Zhang, J.G. Luo, L. Zhao, S.Q. Yang *A Peer-to-Peer Network for Live Media Streaming - Using a Push-Pull Approach.* Proc. MM 2005.
21. Y. Chu, S. G. Rao, and H. Zhang *A Case for End System Multicast.* Proc. ACM Sigmetrics 2000.

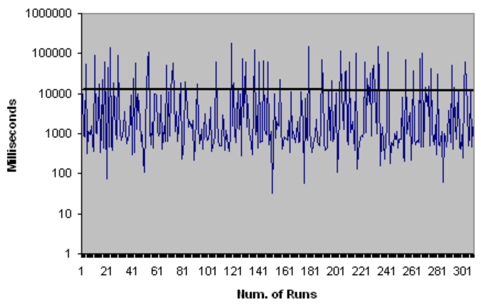
Appendix

In Figure 16, we show the details of each scenario for the Join procedure. Note that, in all the nodes, the algorithms run n times, from the initial joining to the system until the end of the life time. So, the increasing *number of runs* scans the advance of time: this parameter is used in the x-axis. Instead, in the y-axis you can find the time of execution in milliseconds. In the analysis of our experiments, we use a data filter in order to cut off the main spikes. For correctness, this filter is not too strong, so you can see some spikes even in the charts. An interesting aspect of these charts is the presence of a *trend line* (in bold black): this line is the average trend of the time performance and it shows an important quality of the Join procedure. Note that the increasing of the *number of runs* dovetails with the increasing of the time and with the increasing of the overlay network's cardinality. So, the *trend line* demonstrates that the Join procedure (like all the other procedures) is scalable w.r.t. the number of peers. This is a very important result, being StreamComplete realized in order to transmit a video signal from one source to a very large number of users.

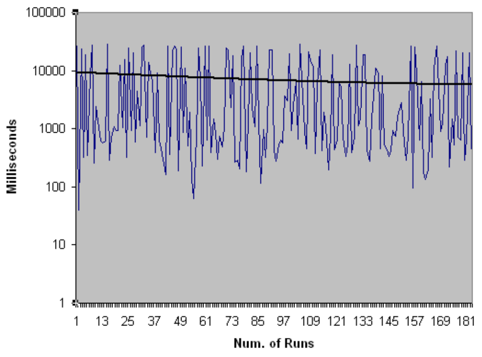
Figure 17 provides details on FastTop: the *trend line* demonstrates that this procedure is scalable w.r.t. the overlay network cardinality. Furthermore, note that Fast Top reacts very well even in case of high churn rate (5th scenario).



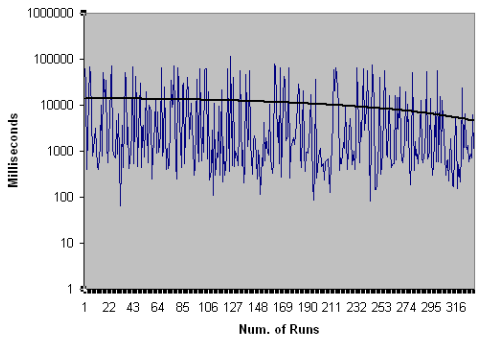
(a) Professional Entertainment



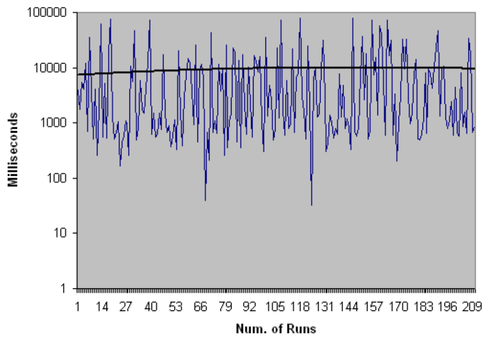
(b) Amatorial Entertainment



(c) Security

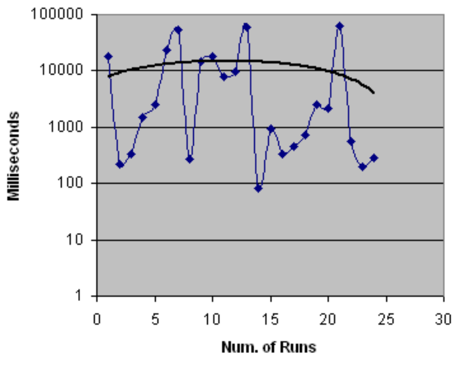


(d) Environmental Monitoring

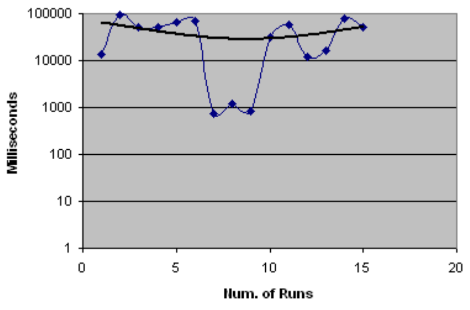


(e) Stress

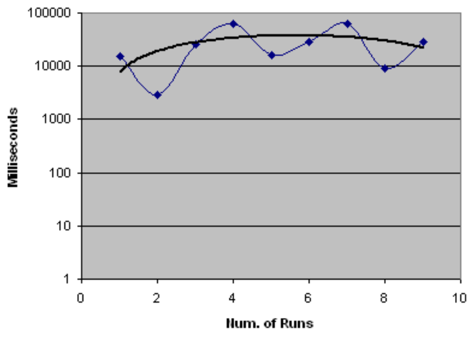
Fig. 16. Join



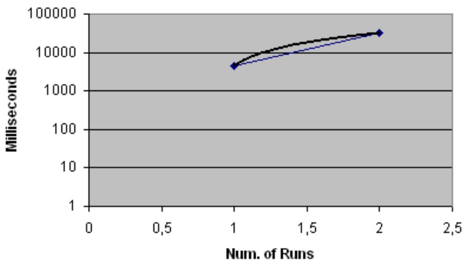
(a) Professional Entertainment



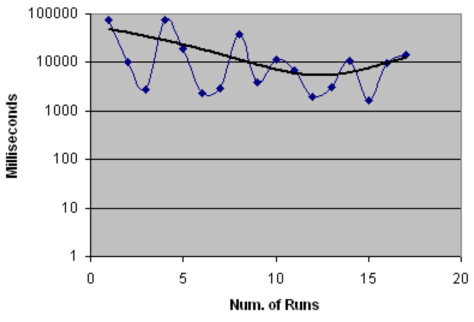
(b) Amatorial Entertainment



(c) Security



(d) Environmental Monitoring



(e) Stress

Fig. 17. Fast Top