StreamComplete: an Architecture for Mesh-based Peer-to-Peer Live Video Streaming

APPENDIX TO THE MAIN PAPER

Federico Covino
DIS - Dipartimento Informatica e Sistemistica Sapienza - Università di Roma
Via Ariosto 25, 00185 Roma (Italy)
Room B213 - Tel.: +39 0677274013 federico.covino@gmail.com

Massimo Mecella
DIS - Dipartimento Informatica e Sistemistica Sapienza - Università di Roma
Via Ariosto 25, 00185 Roma (Italy)
Room B201 - Tel.: +39 0677274028 mecella@dis.uniroma1.it

July 14, 2008
Abstract

This document is the Appendix of the article “StreamComplete: an Architecture for Mesh-based Peer-to-Peer Live Video Streaming”. Here are explained all the details of the architecture, algorithms and testing of StreamComplete. You can find the UML class diagram of the internal architecture, the pseudocode and sequence diagram of the algorithms and, finally, the details of all the results of the testing made in the PlanetLab testbed.

1 Introduction

StreamComplete is a new prototype system for peer-to-peer multimedia streaming. It has been designed in UML and developed in Java. We tested it into the Planetlab testbed with a number of nodes that ranges from 150 to 400 and over, ideally with a maximum of about 800.

This appendix is organized as follows. Section 2 shows UML class diagram of the main layers of StreamComplete. Section 3 presents the pseudocode and the sequence diagram of the algorithms. In Section 4 the reader can find all the details of the tests.

2 Architecture & UML

![Diagram](image)

Figure 1: The internal architecture of a StreamComplete node

This paragraph shows the design of a StreamComplete node. Its internal architecture consists of three logical layers (see Figure 1): the Network Access Object package (NAO) exchanges control data (using TCP) solving the mapping between Java data structures and TCP control packets; the middle layer consists of Controller and Domain Object (DO) packages, that implements all business logic of StreamComplete; the upper layer is the View package and implements either presentation logic and video stream control logic, using RTP on UDP.
2.1 Package N.A.O.

The Network Access Objects (NAO) package consists of twelve Java classes (see Figure 2):

- **ListenerNAO**: is a multithread TCP Server and starts in the boot fase; for each new connection, this TCP server starts a new thread using **ClientHandler**.

- **ClientHandler**: it is used by **ListenerNAO** to create a new thread that processes any new request received from other peers. In particular, **ClientHandler** recognizes the type of received message and forward it to the appropriate class in package NAO. Note that **ClientHandler** has a reference to an instance of **Node** that represent the current peer (referred as *me*).

- **NAO classes of all the procedures**: **DepartureNAO**, **ElectionProcedureNAO**, **FastTopNAO**, **JoinNAO**, **LoopAvoidanceNAO**, **MembershipNAO** and **SchedulerNAO**; all these classes provide static methods for sending and receiving particular messages in the form of TCP packets. For demonstration purposes, we show the class **FastTopNAO** in Figure 6.

- **UtilityNAO**: (see Figure 3) this class provides two important methods to perform the mapping between TCP packets and Mex objects. The first method, **createInfoToSend()**, performs the mapping from Mex objects to TCP packets; the second, **createMexFromString()**, performs the mapping in the opposite direction.

All the classes maintain the reference to the **Node** *me*, created in the boot fase, wich represents the abstraction of the current peer.

2.2 Package D.O.

The Domain Objects (DO) package consists of three Java classes (see Figure 8):

- **VideoSource** (also called **Server**): this 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 references to a **VideoSource** object, to its own local view (that is 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 Node object, plus two flags to know if that node is a supplier or a child or none of them and the identifier of the possible video stream (from/to that node).
2.3 Package Controller

The package Controller consists of twenty-eight classes: seven implement the procedures of StreamComplete; nineteen implement the specific abstractions of the TCP control packets (extending the class Mex); the last two classes use the pCap framework\(^1\) to analyze the incoming and outgoing IP traffic. In particular (see Figure 5):

- **EthernetPacketsCounter_V2**: this is the java wrapper of the network adapter; through pCap, you can implement a single wrapper for all the network adapters (wifi, ethernet, bluetooth etc.) of the node;

- **BandwidthAnalyzer**: is used by Node objects and provides the methods to analyze the incoming and outgoing IP traffic;

- the “*-Mex” classes: they 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. So, almost all of the messages are made only from header and, for this reason, the control traffic overhead in StreamComplete is very low (see Section 4).

- StreamComplete’s procedures (see Figure 7): each Java class provides methods to start the procedure or to reply to an external request. In fact, all the procedures are distributed and require interaction between two or more peers.

\(^1\)See http://www.tcpdump.org/
### Figure 2: TCP multithread server

```java
ClientHandler
Attributes
package Socket connectionSocket
package BufferedReader inFromClient
package DataOutputStream outToClient

Operations
public ClientHandler( Socket clientSocket )
public void run( )

0, *

currentNode

1

Node
{ From DO }
Attributes

ListenerNAO
{ From NAO }
Attributes

Operations
public ListenerNAO( Node nodeMe, int port )
public Node getCurrentNode( )
public void setCurrentNode( Node currentNode )
```

### Figure 3: UtilityNAO class

```java
public String PIPE = "1"
```

```java
public void closeResources( BufferedReader in, BufferedReader out, Socket senderSocket, ServerSocket ss )
private UtilityNAO( )
public String createInfoToSend( Mex mex )
public Mex createMexFromString( Class type, String info )
```

### Figure 4: Controller.Mex.Mex class

```java
public Mex( )
public Mex( Node sender, Node receiver )
public Node getReceiver( )
public void setReceiver( Node receiver )
public Node getSender( )
public void setSender( Node sender )
```
Figure 5: Software module for IP traffic analysis

Figure 6: FastTopNAO class
Figure 7: The procedures of StreamComplete
3 Algorithms and Protocols

This Section describes in detail the Join, Fast Top and Loop Check procedures. For each algorithm is provided a description in natural language and the pseudo-code.

3.1 Join Procedure

In only few seconds a new node can join the overlay network and receive the video stream. 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.

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 and this is an important feature especially in scenarios of emergency or safety.

Being the Join procedure parametric w.r.t. to 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. So, in this case, 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 avoid delays.

The representation in Algorithm 1 (Join) summarizes the major steps of the procedure.
Algorithm 1 Join

Input: currentNode: Node; videoSource: Node;
Output: void

(1) upon event [INIT]:
   JoinReqMex jm;
   jm.sendTo(videoSource);

(6) upon event [JoinReqMex jrm DELIVER | originalSender, sender]:
   if(currentNode.isTheVideoSource())
      if(available_input_bandwidth < x% of nominal_input_bandwidth):
         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);
   else
      if(available_input_bandwidth < x% of nominal_input_bandwidth):
         Node first = first best Supplier with high hop number;
         jrm.forwardTo(first);
      else
         currentNode.children ∪ = {originalSender};
         StartOkMex som;
         som.sendTo(originalSender);

(27) upon event [StartServerOkMex DELIVER | videoSource]:
   currentNode.suppliers ∪ = {videoSource};
   currentNode.hopNumber = 1;

(30) upon event [StartServerNotOkMex DELIVER | videoSource]:
   STOP;

(32) upon event [StartOkMex DELIVER | sender]:
   currentNode.suppliers ∪ = {sender};
   currentNode.hopNumber = min {x | x = currentNode.suppliers[i].hopNumber ∀ i} + 1;
3.2 Fast-Top Procedure

The main innovation of StreamComplete consists in the merge of the tree- and mesh-based system best practices. The Fast-Top procedure derives from this particular merge and gives a smart behavior to the network: to increase the general performance of the overlay network, it is important to avoid the bandwidth bottleneck. Suppose that the state of health of a peer $X$ is much greater than its supplier $Y$: the network can have beneficial effects if we reduce the hops number of $X$, that is, if we place $X$ in the internal area of the network, nearer to the video source, through an exchange of positions between $X$ and $Y$. In fact, you can see a P2P network like a power and signal amplifier: it is clear that the worst nodes must place themselves in the peripheral areas while, on the contrary, the best ones must trace the network back to the video source and back to the central areas. In a mesh graph, swapping the position of two nodes is very difficult: while in a tree there is only one father for each node (excluding root), in a mesh graph the links are very complex and a swapping procedure must manage this complexity (see Figure 10).

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 $Y$ peers is very difficult because 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 churns. So this first solution is not good and must be avoided.

B. There is another most efficient solution that does not need transactions and that allows to reduce the hop number of $X$ node (see figure 11). 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 very low.

StreamComplete uses this second solution through the following six steps procedure called Fast Top:

1. The first step consists in the searching of the lowest-health-state supplier: if the value is lower then a particular percentage (based on the parameter gapPercentage) the current peer can start the procedure sending the FastTopReqMex messages. The potential request’s receivers are ordered by increasing health value and the current peer chooses the first one.

2. On receiving a FastTopReqMex message, a peer checks its current health state to verify the real gap with its child (the starter peer). So, the receiver peer replies with a FastTopResYesMex message, if the gap really exists, otherwise with a FastTopResNotMex.

3. In the first case, when the supplier sends the FastTopResYesMex message, it must stop all the video streams received from its own suppliers. So, it sends StopMexByFastTop messages (see the third step in the Figure 11) and all the streams will be stopped.
4. The procedure’s starter peer (see peer X in Figure 11) can receives two different types of response:
   a. a FastTopResNotMex response, that implies only the updating of its own local view and the end of the procedure;
   b. otherwise, the FastTopResYesMex response contains the list of Y’s suppliers: in this case, after updating the local view, the peer X sends a StartMex messages to them (see step fourth in figure 11) in order to create new incoming video streams.

5. On receiving the StartOkMex messages, peer X can accept the new video streams from its new suppliers.

6. At same time, it must send a StartOkMexFastTop message to peer Y which becomes now a child.

The representation in Algorithm 2 (FastTop) summarizes the major steps of the procedure:
Algorithm 2 FastTop

Input: currentNode: Node; candidate: Node; gap: int;
Output: void

(1) upon event [INIT]:
(2) FastTopReqMex ftreqm;
(3) ftreqm.sendTo(candidate);

(5) upon event [FastTopReqMex DELIVER | child]:
(6) if (currentNode.health >= gap% of child.health)
(7) FastTopResNotMex ftnot;
(8) ftnot.sendTo(child);
(9) else
(10) FastTopResYesMex ftyes;
(11) ftyes.sendTo(child);
(12) foreach Node n ∈ currentNode.suppliers
(13) StopMexByFastTop ftstop;
(14) ftstop.sendTo(n);
(15) currentNode.suppliers = {n};

(16) upon event [FastTopResNotMex DELIVER | sender]:
(17) STOP;

(18) upon event [FastTopResYesMex DELIVER | sender]:
(19) foreach Node n ∈ sender.suppliers
(20) StartMex sm;
(21) sm.sendTo(n);

(22) upon event [StopMexByFastTop DELIVER | sender]:
(23) currentNode.children = {sender};

(24) upon event [StartMex DELIVER | sender]:
(25) currentNode.children ∪ = {sender};
(26) StartOkMex ftok;
(27) ftok.sendTo(sender);

(28) upon event [StartOkMex DELIVER | sender]:
(29) currentNode.suppliers ∪ = {sender};
(30) StartOkMexFastTop ftok;
(31) ftok.sendTo(candidate);

(32) upon event [StartOkMexFastTop DELIVER | sender]:
(33) currentNode.suppliers ∪ = {sender};

13
Figure 10: How to swap the roles of $X$ and $Y$ peers?

Figure 11: The six steps of the Fast Top procedure
3.3 Loop Check Procedure

All the other P2P mesh based video stream systems\(^2\) 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): the system must have a near real time interaction but the main defect of all analyzed systems is the slowness. The loop check mechanism doesn’t slow the system because (i) it is invoked at the end of procedures and (ii) it is a correct but not-complete algorithm, in particular:

- Correctness: a loop check algorithm \( LC \) is correct if, for each graph \( g \), when \( LC(g) = \text{true} \) then \( g \) really contains a cycle;

- Completeness: a loop check algorithm \( LC \) is complete if, given a graph \( g \), when \( g \) contains a cycle then \( LC(g) = \text{true} \);

The correctness is the minimum requirement for every algorithm but in the case of very disadvantageous scenarios, like NP-problems or EXPTIME-problems, the solution can be not complete. A correct and complete loop check algorithm, in case of \( n \) nodes graph, must check all the possible \( i \)-hops paths where \( i = 2, \ldots, n \).

In our particular scenario, it is possible to design a very advantageous correct not-complete algorithm that uses very little bandwidth. Indeed, StreamComplete provides the following two steps procedure:

1. Peer sends to all its children a LoopCheckMex message which contains: the original sender, the current sender, the current receiver, the bound of the possible loops and the TTL. In this first sending, 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 in the lower layer protocol: the package NAO. If is still not been found any loop, the current peer forwards the message to its children with decreased TTL (until it is greater than zero). All this is computed in package NAO in order to increase efficiency. 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 (that is a supplier).

\(^2\) Analyzed in the Article StreamComplete: an Architecture for Mesh-based Peer-to-Peer Live Video Streaming
The Algorithm 3 (LoopCheck) summarizes the major steps of the procedure:

**Algorithm 3 LoopCheck**

<table>
<thead>
<tr>
<th>Input:</th>
<th>currentNode: Node;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>void</td>
</tr>
<tr>
<td>(1)</td>
<td>upon event [INIT]:</td>
</tr>
<tr>
<td>(2)</td>
<td>foreach Node n ∈ currentNode.children</td>
</tr>
<tr>
<td>(3)</td>
<td>LoopCheckMex lcm;</td>
</tr>
<tr>
<td>(4)</td>
<td>lcm.sendTo(n);</td>
</tr>
<tr>
<td>(5)</td>
<td>upon event [LoopCheckMex mex DELIVER</td>
</tr>
<tr>
<td>(6)</td>
<td>if(sender ≠ currentNode )</td>
</tr>
<tr>
<td>(7)</td>
<td>if(mex.TTL &gt; 0)</td>
</tr>
<tr>
<td>(8)</td>
<td>mex.TTL -= 1;</td>
</tr>
<tr>
<td>(9)</td>
<td>foreach Node n ∈ currentNode.children</td>
</tr>
<tr>
<td>(10)</td>
<td>mex.forwardTo(n);</td>
</tr>
<tr>
<td>(11)</td>
<td>else</td>
</tr>
<tr>
<td>(12)</td>
<td>STOP;</td>
</tr>
<tr>
<td>(13)</td>
<td>else</td>
</tr>
<tr>
<td>(14)</td>
<td>currentNode.suppliers − = {sender};</td>
</tr>
<tr>
<td>(15)</td>
<td>STOP;</td>
</tr>
</tbody>
</table>
4 Extensive Experimental Evaluation

In this Section we analyze each algorithm from the point of view of time and generated traffic. We show the data grouped by scenario and grouped by runs, describing the general and particular behavior.

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 framerate: the video frame rate, in a realistic scenario, can range from about 30 Kbps to 1000 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 limits the maximum local-view cardinality in a range from 10 to 100 peers. (4) available bandwidth: in our experiments, each peer can have an ADSL connection that range in the follow set: 2 Mbps, 4 Mbps or 7 Mbps. STREAMCOMPLETE was tested into five realist scenarios, describing in Table 1:

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>Video Frame rate</th>
<th>Churn Rate</th>
<th>LocalView cardinality</th>
<th>Node Bandwidth Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Professional Entertainment</td>
<td>100 Kbps</td>
<td>x=3 w=0 t=2 p=0.4</td>
<td>60</td>
<td>20% @ 2 Mbps, 30% @ 4 Mbps, 50% @ 7 Mbps</td>
</tr>
<tr>
<td>2-Amatorial Entertainment</td>
<td>60 Kbps</td>
<td>x=3 w=0 t=2 p=0.4</td>
<td>100</td>
<td>20% @ 2 Mbps, 30% @ 4 Mbps, 50% @ 7 Mbps</td>
</tr>
<tr>
<td>3-Security</td>
<td>60 Kbps</td>
<td>x=3 w=0 t=2 p=0.3</td>
<td>20</td>
<td>20% @ 2 Mbps, 80% @ 7 Mbps</td>
</tr>
<tr>
<td>4-Environmental Monitoring</td>
<td>50 Kbps</td>
<td>x=3 w=0 t=2 p=0.3</td>
<td>40</td>
<td>50% @ 2 Mbps, 20% @ 7 Mbps</td>
</tr>
<tr>
<td>5-Stress</td>
<td>40 Kbps</td>
<td>x=2 w=0 t=1 p=0.7</td>
<td>100</td>
<td>60% @ 2 Mbps, 40% @ 7 Mbps</td>
</tr>
</tbody>
</table>

Table 1: Testing Scenarios.

The professional entertainment and the amatorial entertainment scenarios have the same churn rate and the same kind of population, but a different video frame rate. The security scenario has a medium video quality, a very low churn rate and only two type
of peers: the peers at 2 Mbps, that represent some legacy device, and the peers at 7 Mbps, that in our testing represent the highest performing nodes. In the environmental monitoring, the majority of the peers has not the fastest connection, in order to reproduce a monitoring of an area not well served. The stress scenario is designed to increase the churn rate and so to increase the dynamic changes of network topology.

We totally collected data on 22,136 runs of the algorithms with 584,823 messages exchanged between the nodes. In particular:

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>NUM. of RUNS</th>
<th>NUM. of MESSAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Professional Entertainment</td>
<td>6.604</td>
<td>144,571</td>
</tr>
<tr>
<td>2 - Amatorial Entertainment</td>
<td>4.460</td>
<td>121,527</td>
</tr>
<tr>
<td>3 - Security</td>
<td>2.907</td>
<td>70,217</td>
</tr>
<tr>
<td>4 - Environmental Monitoring</td>
<td>4.215</td>
<td>142,777</td>
</tr>
<tr>
<td>5 - Stress</td>
<td>3.950</td>
<td>105,731</td>
</tr>
</tbody>
</table>

Table 2: Data collected during tests

4.1 Testing the Join Procedure

Each node is able to connect itself to the overlay network in only few seconds, in a completely decentralized way. As shown in Figure 12, in average, each peer needs from 5.2 to 15.5 seconds to connect itself to the overlay network and receive immediately the video stream. This is an excellent result, even comparing it to the actual state of art. In Table 3, the details of the performances grouped by scenario are shown.

In Figure 14, we show the details of each scenario. 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):
Table 3: Join Performance, grouped by scenario

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>Average Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Professional Entertainment</td>
<td>5.19</td>
</tr>
<tr>
<td>2 - Amatorial Entertainment</td>
<td>13.36</td>
</tr>
<tr>
<td>3 - Security</td>
<td>7.05</td>
</tr>
<tr>
<td>4 - Environmental Monitoring</td>
<td>15.52</td>
</tr>
<tr>
<td>5 - Stress</td>
<td>9.90</td>
</tr>
</tbody>
</table>

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.

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 13 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.

Figure 13: Average Join control traffic
Figure 14: Details of the Join performance for each scenario
4.2 Testing the Fast-Top Procedure

Fast Top is a very complex procedure that involves, in each run, a set of peers that agree on a particular overlay re-configuration. Figure 15 shows the time performance through the bar chart (relative scale on the left) and the average size of a Fast-Top message through the line chart (relative scale on the right).

![Figure 15: Performance of Fast Top grouped by scenario](image)

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>Average Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Professional Entertainment</td>
<td>11,693</td>
</tr>
<tr>
<td>2 - Amatorial Entertainment</td>
<td>39,953</td>
</tr>
<tr>
<td>3 - Security</td>
<td>27,932</td>
</tr>
<tr>
<td>4 - Environmental Monitoring</td>
<td>18,097</td>
</tr>
<tr>
<td>5 - Stress</td>
<td>16,984</td>
</tr>
</tbody>
</table>

Table 4: Fast Top Performance, grouped by scenario

The shown data include the time of disconnection and re-connection of all the involved peers, so they represent an excellent result. Obviously, the Fast Top procedure is invoked only in particular network configuration and, for this reason, the number of runs is directly proportional to the local view cardinality of the scenarios. Even in this case (see Figure 17), 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).
Another import result of our experiments is the very low average control traffic of Fast Top. This complex procedure, in average, takes at most 0.56 KB per run. 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.
Figure 17: Details of the Fast Top performance for each scenario
4.3 Testing the Loop Check Procedure

The Loop Check procedure represents one of the main innovations of StreamComplete, designed in order to avoid unnecessary use of bandwidth: in a live video streaming context, a loop in the signal routing causes waste of resources and additional delays. In this Section we analyze two 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.

Figure 18: Loop Check performance, grouped by scenario

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>Average hops number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Professional Entertainment</td>
<td>3.44</td>
</tr>
<tr>
<td>2 - Amatorial Entertainment</td>
<td>2.69</td>
</tr>
<tr>
<td>3 - Security</td>
<td>3.55</td>
</tr>
<tr>
<td>4 - Environmental Monitoring</td>
<td>2.61</td>
</tr>
<tr>
<td>5 - Stress</td>
<td>5.87</td>
</tr>
</tbody>
</table>

Table 5: Loop Check Performance, grouped by scenario

The results shown above are directly proportional to the configuration of the scenarios. For example, look 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 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 19 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 indicates that StreamComplete can organize the mesh graph in compliance with two main properties of the small worlds network\(^3\): (1) low average hop distance between any two randomly chosen nodes and (2)

\(^3\)The notion of small world phenomenon originates from social science research. It has developed to become a very active current research topic in physics, computer science, and mathematics. It has been observed that the small world phenomenon is pervasive in a wide range of settings such as social communities, biological environments, and data-communication networks.
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 (for example, in a flash crowd scenario). Informally, 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. This property implies that one can find information stored at any random node of a small world network by only a small number of link traversals. In the video streaming context, a small worlds network avoids the formation of long chains of forwarding and this is very important in order to reduce delays.

From the point of view of generated traffic, Figure 20 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.