# Live Streaming on a Peer-to-Peer Overlay: Implementation and Validation

Joaquín Caraballo Moreno[1] and Olivier Fourmaux[2]

[1] Universidad de Las Palmas de Gran Canaria**
[2] Laboratoire d'Informatique de Paris 6 (CNRS / Université Pierre et Marie Curie)

**Abstract.** Peer-to-peer based applications have demonstrated their interest in specific fields like file-sharing or large-scale distributed computing. Few works study the viability of peer-to-peer based continuous media applications. In this context, we get concerned with developing a framework that provides a service of transmission of a single source data flow with the characteristics and requirements of non-interactive audio and video live streaming. To perform experimentation, we have developed an application that uses a simple peer-to-peer protocol that can be easily changed, and can be used as a framework to test different protocols to build and maintain overlay network. We have carried out some tests and measurements in an hybrid approach to evaluate the quality of the application with its protocol and to offer a base to the evaluation of future protocols.

## 1 Introduction

Nowadays popular radio stations offer the possibility of retrieving their emission over the Internet. However, if the public massively used this service, required resources would be huge. Also, when some program is especially requested, if available resources are not heavily oversized it will not be possible to offer a quality emission to all listeners. With an application like ours, where user nodes also act as relayers, a solution to those difficulties could be provided.

In another scenario, a peer-to-peer application-level multicast approach enables anyone with an ordinary Internet connection to broadcast to potentially any number of listeners. Such a tool could contribute to the use of Internet as a communication media from anyone to many others, and, therefore, assisting to develop the right to free speech.

Radio and video live streaming from a single source to multiple receivers is an area of growing interest that poses some difficulties. With non-interactive streaming, latency constraints are softened but jitter constraints remain. In this work we have concentrated on the issues of efficient data transmission from one sender to a high number of receivers, addressed to non-interactive radio or video live streaming.

** This work has been carried out during an internship at LIP6.

P. Lorenz and P. Dini (Eds.): ICN 2005, LNCS 3421, pp. 625–633, 2005.

The traditional approach uses a different *unicast* data stream between the sender and every receiver. As lower layers does not use the fact that the data to send are the same for every receiver, the required resources of broadcasting to $N$ receivers are $N$ times those of sending to one of the receivers. Also processing and network resources will have to be proportioned to the number of receivers.

Some standards have been developed providing a *multicast* service, At network layer, some additions to IP have been thought up, and the new IPv6 includes them. Also, at transport layer several protocols have been proposed. However no protocol have been really deployed for general use.

Application-level multicast offers a certain efficiency for one-to-many transmissions keeping lower layer protocols unchanged. Even if remaining out of underlying layers imposes us certain limitations in efficiency, application-level multicast provides us with several possibilities. On the one hand, it can be used to put in practice some multicast techniques and improve them. On the other hand, it permits to develop and use applications that require and can take advantage of a multicast service. Our work uses this kind of approach.

To develop an application-level multicast service, some agents must cooperate relaying received data to others. We have chosen a non-hierarchical approach, where every node has the same category, being a client when requesting the data flow and being a server when retransmitting the data flow to the nodes that require it. That is, we use a *peer-to-peer* network to put into practice the multicast service. All nodes have the same behavior except for the root, which, instead of requesting the data flow to relay it, provides the data flow itself and introduces it at first into the network. Many peer to peer networks have been developed where users transmit and share files, that is, static content; but what we are transmitting is a data flow, whit dynamic nature and time constraints.

The rest of this paper is organized as follows. Section 2 outlines our framework, decribing the service provided by our application, the underlying protocol and the hybrid approach we use for our experimentation. The results are proposed in section 3. Section 4 describes related works and section 5 concludes.

## 2 Live Streaming Framework

### 2.1 Service Offered by the Application

The transport service provided by the application is transparent to its users, letting them to be completely unaware of how the flow is being sent. At the sender side, a source connects to a node and sends through it a data flow. After the other nodes, a sink obtains its flow from the node to which it is connected. Figure 1 illustrates this service model.

### 2.2 A Simple Protocol

We have kept protocol definition as independent as possible from the remainder of the software. Next we describe the one we have developed and included.

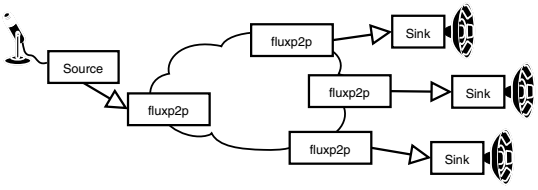


**Fig. 1.** Offered service

Inter-node communication is made by means of a peer-to-peer protocol. For every broadcasting network, one node will act as root, being the primary source of the data flow, while the others will receive it and possibly retransmit it.

To build the delivery tree five different control packets are used. A *Request Packet* is sent by a node who wants to join the network to the node to which it wants to be connected. If the latter can accept the connection, it answers with an *Accept Packet*, otherwise, a *Forward Packet* can be sent expressing that the node will not accept the connection and informing of other possible candidates. To mesure the quality of the connection with another node, a node can send a *Ping Packet*, that will be answered with a *Pong Packet*, thus, permitting to work out the *round trip time*.

**Joining Process.** Joining process is illustrated in Fig. 2. When a node wants to join the broadcast network, only the root is in its candidates list ($C$ in the figure). At first, it sends a *Request Packet* to the root node. Usually, the root will not have any free slot, so it will answer with a *Forward Packet*, indicating that it is not able to directly treat the connection and informing how to reach its direct descendants. Consequently, the node will discard the root node and will add to $C$ those received within the *Forward Packet*. Next, it sends *Ping Packets*, that will be probably answered with *Pong Packets*, to measure the quality of the candidates. In accordance with those RTTs, a new *Request Packet* will be sent

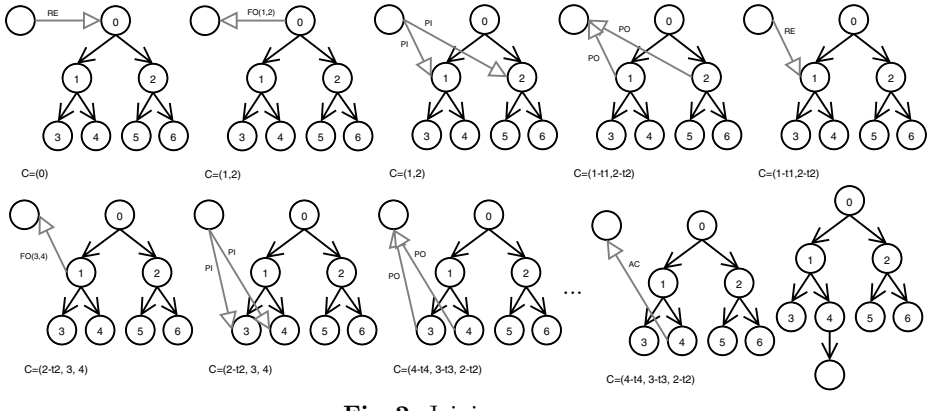


**Fig. 2.** Joining process

to the best candidate. Again, if it has no free slot, it will answer with a *Forward Packet* and the process will still continue up to a node with a free slot will be contacted and answer with an *Accept Packet*, finishing the joining process.

**Scalability Issues.** The protocol has been developed to test the software architecture with a maximum of simplicity, but also trying to keep the scalability of the system. In a strict sense, as the proposed joining procedure always starts by contacting the root, if the number of simultaneous joining nodes is very high, the root will became overcharged and therefore the system does not really scale up. However, instead of making every node to start by the same root, we can set, in the non-root nodes configuration, different nodes as root nodes.

### 2.3 Hybrid Prototype

To evaluate the quality of a protocol or a flow broadcasting application in general, the deployment of a big amount of nodes is necessary. In an ideal situation, we should have a host available for each node. However, as the desired number of nodes for a meaningful test is considerably high, in most of situations such a number of computers will not be available. As a workable solution, we can deploy the set of $N$ nodes on $M$ hosts, where $N >> M$. But if those nodes are located in the same machine or in machines close together, the experienced characteristics of network connections will not be realistic. To deal with this difference we have used a network topology model, composed of a set of nodes and delays between any two of them. For every node in the broadcasting network, we assign to it a node in the topology model, enabling us to obtain a simulated network delay between any pair of nodes. Whenever we receive a packet we simulate its corresponding network delay by waiting before we treat it. This system enables us to really test a broadcast network of a considerably high number of nodes without having to deploy them in a so big number of hosts.

## 3 Experimental Results

### 3.1 100 Nodes Joining Together

In this setting we have deployed a network of 101 nodes, where every fifth second one non-root node joins. 10 minutes after start time, we have stopped them all. All nodes where deployed on the same machine. During the experiment, the root has been sending a 40 bytes packet every 100 miliseconds.

**Latency.** Our experiments have been based on measuring latency, considered as the time interval between the emission of a packet, just before it enters into the root node, and the reception of the packet at a receiver side, just after it go out from the corresponding non-root node. Theoretically, this time interval will be the result of $latency = \text{processing}(n_0) + \sum_{i=1}^{r}(\text{link}(n_{i-1}, n_i) + \text{processing}(n_i))$, where $n_0$ is the root node, $n_r$ the reception node, $n_i$ a node in the way from the root to the reception node, $\text{processing}(n_i)$ the processing time of the flow in $n_i$, and $\text{link}(n_i, n_j)$ the network delay between nodes $n_i$ and $n_j$. In our experiments

we have simulated network delays, and, thus, also link times are produced in the host and processing times are incremented because of the cost of this simulation.

In Fig. 3, we can watch how latency is growing as the number of nodes receiving packets is incremented. To study performance, we have taken an arbitrary node, we have taken an arbitrary node, *node 0*, and observed how the latency experienced evolves. In Fig. 4 we can watch this evolution related with the number of nodes receiving data.

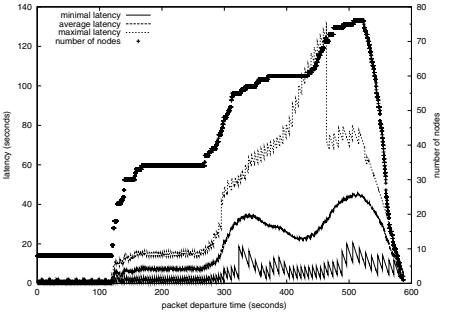


**Fig. 3.** Minimal, average and maximal latency with number of nodes (*to-100*)

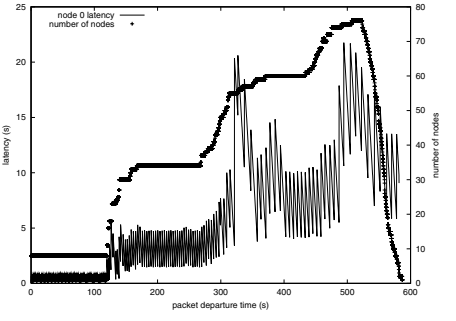


**Fig. 4.** Node 0 latency with number of nodes (*to-100*)

We can observe that periodically latency is abruptly incremented to quickly recover its usual lower level. Figures 5 and 6 show this behavior in detail. Between 0 and 10 seconds, we can see this fast peeks, that surge from much lower values. At 200 seconds peeks have become greater and involve more blocked packets.

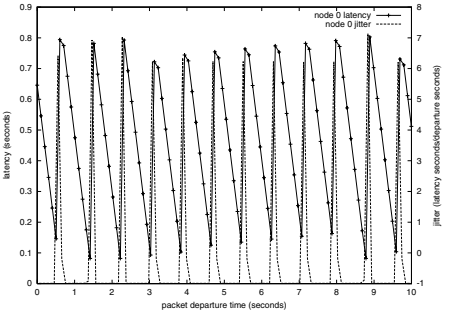


**Fig. 5.** Node 0 latency and jitter between 0 and 10 seconds (*to-100*)

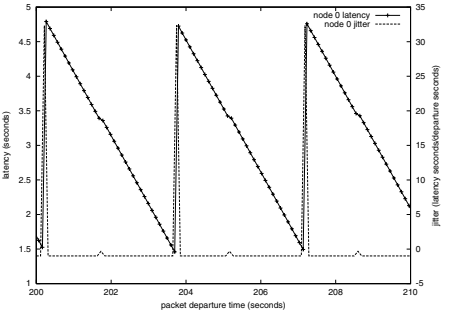


**Fig. 6.** Node 0 latency and jitter between 200 and 210 seconds (*to-100*)

We think these peeks are caused by the way in which incoming packets are treated. Nodes are permanently waiting for incoming packets. But, as the system does not notify fast enough their arrival, when a node knows about them there

are already several ones waiting to be treated. Those packets are treated, but the first has already waited a lot, the second a bit less, etc., until the last of them which almost has just arrived and so its delay is small. Once it has not other packets to treat, the node come back to wait for new packets and the initiative is passed to the operating system. It will not be notified about new packets until a certain quantity of them have arrived, and the same problem is repeated again.

The size of those jumps start about 0.7 seconds and goes to 5 seconds and further. As this component is much greater than network delays, we deduce that, in general, latencies in the broadcasting network will be basically determined by the processing capacity—or difficulty—of node hosts.

**Jitter.** Another important characteristic in live streaming is jitter, that is, the variation of latency. Again, we start with the theoretical measurement: $jitter = \partial latency/\partial time$, that is, the jitter is the derivative of the latency with respect to time. In practice, we just divide latency and time differences between a received packet and the next one: $jitter = \triangle latency/\triangle time$.

In Figs. 7 and 8, we can see how there are constantly some very high peeks whether most of time relative increment is slightly under 0. Again, those latency peeks we talked about are causing the behavior of the jitter. Detailed graphs in Figs. 5 and 6 also show the jitter with its originating latency.

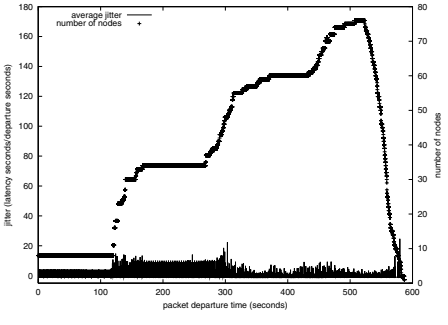


**Fig. 7.** Average jitter with number of nodes (*to-100*)

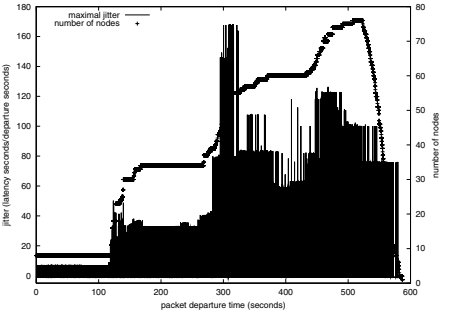


**Fig. 8.** Maximal jitter with number of nodes (*to-100*)

### 3.2 10 Nodes Joining Together

In this new setting 11 nodes have been deployed. Like before, 40 bytes data packets are sent by the root every 100 milliseconds for 10 minutes and all nodes, including root, are executed on the same machine.

We observe that all the 10 receivers are receiving packets during almost all the experiment and the resulting latency measures are more or less stable. Again the zigzag effect can be observed. We can watch in the detailed graph, in Fig. 9, referred to nodes 4, 3, 7, and 5 for the first 10 seconds, how the same pattern is repeated, keeping a jump size approximately of one second. Besides, we can

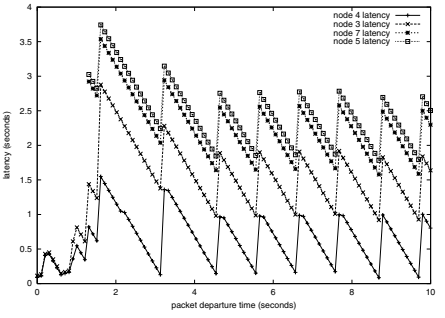


**Fig. 9.** Nodes 4, 3, 7, and 5 latency between 0 and 10 seconds (*to-10*)

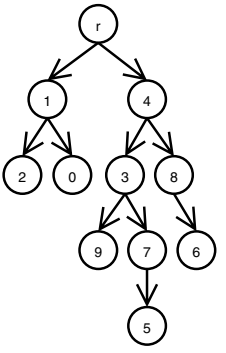


**Fig. 10.** Resulting broadcast network topology (*to-10*)

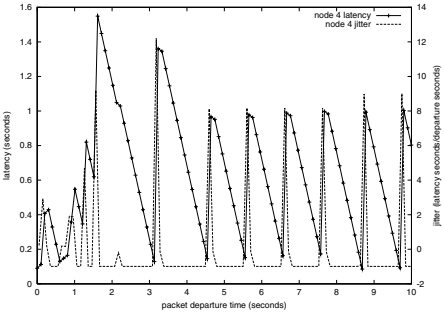


**Fig. 11.** Node 4 latency and jitter between 0 and 10 seconds (*to-10*)

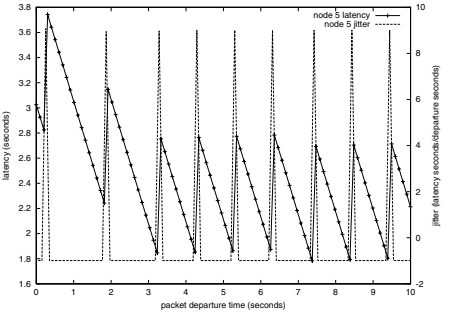


**Fig. 12.** Node 5 latency and jitter between 0 and 10 seconds (*to-10*)

also observe that jumps are practically simultaneous in the different nodes, as this effect is produced by the system that they share.

Figure 10 shows the topology created by the joining process. Nodes 4, 3, 7 and 5 form a path from root to a leaf. Looking again at Fig. 9, we observe how the interval between the latency experienced by a node and the latency experienced by its descendant stands stable. For every pair of nodes it stands with a fixed value $l_{i,j}$ , where $0.1\,\mathrm{s} < l_{i,j} < 1\,\mathrm{s}$ . The most interesting of this jitter is that it will allow us to size the reception buffer quite accurately.

## 4 Related Work

A first article from Bawa, Deshpande and Garcia-Molina [1] focus on streaming media application with peer-to-peer approach. Their work result in an real application[2] that permits to multicast an audio flow to a number of receivers. It implements a peer-to-peer application-level multicast approach and uses the

Gnutella protocol[3] to find appropriates nodes. Before introducing streaming over peer-to-peer, End System Multicast[4] propose an alternative to IP multicast with an application level overlay. Overcast[5] is another application-level multicasting system that could handle streaming but like the previous proposition, it relies on a long-lived infrastructure of dedicate hosts. Other important works dealing with streaming over peer-to-peer: ZIGZAG[6] relies on a complex clustering system; PROMISE [7] insists on the many-to-one scheme; CoopNet[8] is fully tied with the flow semantic; GnuStream[9] is a streaming layer to use over another peer-to-peer infrastructure; and SplitStream[10] distribution of the flows over all the participant but must take place on a structured peer-to-peer architecture. The feasibility of large-scale peer-to-peer live streaming is analysed with network traffic in [11].

## 5 Conclusion

We have achieved a framework that will permit to easily develop and test a flow broadcasting protocol. Protocol developing will be freed of other application building details. Resulting applications will have a testing scenario at its disposal where a high number of nodes can be deployed on a small number of hosts. When measuring the protocol efficiency we can see that a good stability can be reached with a reasonable number of nodes per host. The main contribution to latency and jitter is the zigzag effect, produced by the system and increased by its load.

New protocols should be developed, integrated in the software and tested. Such protocols should take into account possible network variations and have a dynamic vision of the network. Our application transparently gives a service of transmission to multiple receivers. Neither sender nor receivers are concerned about how we carry it out. However, sometimes is interesting to allow ends to exercise some kind of regulation. Therefore, we could modify the software architecture and the protocol to dynamically respond to user needs.

This work results of an internship continued the work *Distribution pair à pair de flux audio* carried out at LIP6 by the autors and Alejandro Rodríguez San José. Sources and documentation are available as a GPL Savannah non-gnu project at `http://savannah.nongnu.org/projects/fluxp2p`. See `http://savannah.nongnu.org/cvs/?group=fluxp2p` to retrieve it from CVS.

## References

1. Bawa, M., Deshpande, H., Garcia-Molina, H.: Transience of peers and streaming media. HotNets-I (2002)
2. : peercast.org. Website (accessed in 2004) `http://www.peercast.org/`.
3. Clip2: The gnutella protocol specification v0.4 (accessed in 2004) `http://rfc-gnutella.sourceforge.net/developer/stable/index.html`
4. Chu, Y.H., Rao, S.G., Zhang, H.: A case for end system multicast. In: Measurement and Modeling of Computer Systems. (2000)

5. Janotti, J., Gifford, D.K., Johnson, K.L., Kaashoek, M.F., O'Toole, Jr., J.W.: Overcast: Reliable multicasting with an overlay network. In: USENIX OSDI. (2000)
6. Tran, D., Hua, K., Do, T.: Zigzag: An efficient peer-to-peer scheme for media streaming. In: IEEE INFOCOM. (2003)
7. Hefeeda, M., Habib, A., Boyan, B., Xu, D., Bhargava, B.: Promise: peer-to-peer media streaming using collectcast. Technical Report CS-TR 03-016, Purdue University (2003)
8. Padmanabhan, V., Wang, H., Chou, P., Sripanidkulchai, K.: Distributing streaming media content using cooperative networking. ACM/IEEE NOSSDAV (2002)
9. Jiang, X., Dong, Y., Xu, D., Bhargava, B.: Gnustream: A p2p media streaming system prototype. In: International Conference on Multimedia and Expo. (2003)
10. Castro, M., Druschel, P., Kermarrec, A., Nandi, A., Rowstron, A., Singh, A.: Splitstream: High-bandwidth multicast in cooperative environments. ACM Symposium on Operating Systems Principles (2003)
11. Sripanidkulchai, K., Ganjam, A., Maggs, B., Zang, H.: The feasibility of supporting large-scale live streaming applications with dynamic application end-points. ACM SIGCOMM (2004)