NeuroCast: Adaptive Multi-Source P2P Video Streaming Application for wireless networks

Carlos Gañán, Juan Caubet, Sergi Reñé
Jorge Mata-Díaz, Juan J. Alins and Óscar Esparza

Universitat Politècnica de Catalunya (Departament Enginyeria Telemàtica)
1-3 Jordi Girona, C3 08034 Barcelona (Spain)
{carlos.ganan, juan.caubet, sergi.rene, jmata, jalins, oesparza}@entel.upc.es

Abstract. Streaming consists in distributing media to large audiences over a computer network. Providing a streaming service for wireless mobile nodes presents many challenges. A peer-to-peer (P2P) solution has the big advantage of seamlessly scaling to arbitrary population sizes, as every node that receives the video, while consuming resources, can at the same time offer its own upload bandwidth to serve other nodes. In this paper we present the design and implementation of NeuroCast: an unstructured P2P application for video streaming. NeuroCast implements a robust scheduling algorithm which minimizes the scheduling delay. Moreover, given heterogeneous contents, delays and bandwidths. Thus, NeuroCast becomes suitable for wireless scenarios due to its capability to adapt to changing network conditions.

Keywords: Video streaming, P2P networks, multi-source, wireless

1 Introduction

During the last decade, the growth of popular web sites serving multimedia contents has led to the increase of video streaming applications. However, video streaming over a wireless network has to deal with several challenges: 1) nothing is guaranteed about bandwidth, delay, and packet loss rate; 2) it is difficult to predict the bandwidth, delay, and loss rate information, since it is unknown and time-varying; 3) the heterogeneity of receiver capabilities is a significant problem when video streams are distributed over a multicast network; and 4) a congestion/flow control mechanism has to be employed to prevent congestion epochs in the wireless network.

There have been proposed several ways to approach these challenges. The traditional client-server model is suitable for streaming, but it presents scalability problems and a loss of efficiency in resource exploitation. In fact, a server has a limit bandwidth and cannot serve more than a limited number of clients at the same time. The best way to distribute multimedia content from a source to a group of hosts at the same time is the IP multicast. However the deployment of IP multicast has been limited due to several reasons. First of all it requires changes in the network devices increasing the complexity and the overhead at the routers. But also, it presents commercial problems

since a lot ISPs disable it. Another possibility is to use peer-to-peer networks for the streaming. In these networks, the stream receivers act as clients and servers at the same time (i.e. servent) replicating packets they receive. The aim for these techniques is to allow bandwidth-consuming streaming media to be delivered to a large number of consumers in a scalable, robust and efficient manner. In this sense, we present NeuroCast, a P2P open-source solution to video streaming over wireless networks.

P2P streaming systems strive to optimize three important metrics: i) start-up delay, ii) end-to-end delay, and iii) playback continuity index. Most of the systems may be classified based on the type of distribution graph they implement: mainly tree and mesh, though a lot of hybrid solutions have been implemented already. Tree-based overlays implement a tree distribution graph, rooted at the source of the content [1–4]. In principle, each node receives data from a parent node, which may be the source or a peer. If peers do not change too frequently, such a system requires little overhead; in fact, packets can be forwarded from node to node without the need for extra messages. However, in high churn environments (i.e. fast turnover of peers in the tree), the tree must be continuously destroyed and rebuilt, a process that requires considerable control message overhead. As a side effect, nodes must buffer data for at least the time required to repair the tree, in order to avoid packet loss. Mesh-based overlays [5–7] implement a mesh distribution graph, where each node contacts a subset of peers to obtain a number of chunks. Every node needs to know which chunks are owned by its peers and explicitly pulls the chunks it needs. This type of scheme involves overhead, due in part to the exchange of buffer maps between nodes (nodes advertise the set of chunks they own) and in part to the pull process (each node sends a request in order to receive the chunks). Thanks to the fact that each node relies on multiple peers to retrieve content, mesh based systems offer good resilience to node failures. On the negative side, they require large buffers to support the chunk pull, as large buffers are needed to increase the chances of finding the missing chunks in the playback sequence. In this sense, NeuroCast evolves the Peercast [2] tree network to a mesh-based overlay.

Peercast is an open source streaming media multicast tool which is generally used for streaming audio and makes use of a bandwidth distributing approach where users can choose the relay to connect in the downstream. NeuroCast extends Peercast in order to enhance its capabilities making it possible to watch videos even in a wireless scenario. Moreover, NeuroCast adds the necessary design to allow any user to become a *broadcaster*. Among the different improvements made in NeuroCast, it is worth to mention the multi-source streaming capability. In contrast to Peercast, NeuroCast solves the typical asymmetry of the link by using multiple sources. Therefore, it allows to video stream over a wireless network.

In this paper, we first approach the problem of media streaming from a practical point of view. We present the design of NeuroCast, a P2P streaming for wireless networks, that: 1) supports very high levels of churn, 2) supports strongly heterogeneous distributions of peer upload capacity, 3) has multi-source capabilities to use efficiently the available upload capacity, and 4) employs fast adaptation and recovery from abrupt changes in network conditions. Due to this fast adaptation, NeuroCast can operate in a wireless scenario where nodes are prone to suffer from disconnections.

The rest of the paper is structured as follows. Section 2 introduces the basics of Peercast. Section 3 describes the NeuroCast system. Section 4 presents a analysis of NeuroCast, carried out by way of emulation on medium-scale network testbeds. Finally, section 5 concludes this work.

2 Peercast

The multimedia streaming application Peercast is a multi-thread application, so that different processes can be executed at the same time concurrently. The number of processes in Peercast is variable and depends on the amount of connections that are established. The main thread deals with the creation of its child threads. In addition to these tasks, this thread becomes passive waiting for incoming connections. In general, for each request that it receives, a thread is created which will serve the request till its death.

2.1 PCP: Packet Chain Protocol

PCP is the protocol used in Peercast to allow communication among different clients. The main goal of this protocol is to reuse the same data-flow which is used to send the multimedia information, to send the control information too. PCP chaining allows users to 'piggy back' a download from another user.

The first packet of a group indicates the type of the packets that are being sent and the number of packets that come after it. These packets that indicate the type of the group are called *parent packets*. The great potential of this protocol lies in the way the packets are chained, as each one of the packets inside a group can be at the same time parent of another type of packets.

2.2 Entities

Peercast uses the Oriented Object programming, i.e., it has several classes that represent the different system entities. Therefore, Peercast consists of *servent*, *channel*, *buffer* and *root* classes.

Servent. This class becomes essential for the performance of the Peercast application. As any real-time P2P application, the application depends on the network conditions, so the packets can arrive out of order. Hence, it is basic to have a good system to order and store packets efficiently. The servent class manages the channel transmission and listens to requests coming from the network or from the same host. Servents exert the server and the client at the same time. Servents are classified according to the transmission type. Thus, the most used servents are:

- `Server:` It is the server of the main thread which handles any request. This subclass creates new server classes to manage the different requests.
- `Relay:` This class manages the retransmission of a channel among different NeuroCast instances. In this mode, the data are sent using the PCP protocol.
- `Direct:` This class is used to retransmit the multimedia file directly without packing the data. This class is used to handle the packet transmission to a player.

Channel. A channel is originated when a user starts to retransmit new multimedia information to the Peercast network. This user, the first one in uploading the channel to the network, is named the *emitter* or *broadcaster*, and it becomes the unique contact between the original source and the Peercast network. The channel is shared among the different peers.

Buffers. It is obvious the necessity of a buffer in Peercast due to the fact that each node can perform as client as well as server, so it needs to store in memory the information in order to be capable of retransmitting it. Without a buffer, a node can only reproduce the received packets in a player but cannot send them to any other client in the network.

Root Nodes. Despite the fact that Peercast was designed to run in a decentralize way, the actual situation is centralized. The developers implemented Peercast in order to create a network without servers. Thus, Peercast only has clients. If a group of users want to share a video among them, they only need to know the IP address of some of them and the channel identifier. Hence, they will create a download tree. On the other hand, if a user wants a video but it does not know any user from the sharing group, that user cannot see the video. Therefore, this situation makes the tree model of Peercast unfeasible for real scenarios. Peercast deals with this issue with the concept of *root nodes*. 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. Root nodes gather information about other clients in order to create an information directory.

3 NeuroCast

Analyzing the evolution of P2P application for file distribution, it has been noted a leading trend to move from the typical tree/forest topologies to the mesh topology. This evolution allows NeuroCast to use the bandwidth of `N` users which cannot broadcast the video by themselves, but together they are able to achieve the broadcasting. NeuroCast is a multi-thread application, each network user is able to receive a stream from different peers, and at the same retransmit it to any other user.

Users interact with NeuroCast through a web interface quite simple and friendly. By means of these web pages, the user can manage the application as well as get information about the retransmission taking place or about the peers from where the stream is being downloaded.

Apart from the broadcasters who perform as stream sources, there is another type of clients in NeuroCast: the *trackers*. These clients deal with the gathering of information about the peers that are sharing a particular channel. Thus, when a user starts a session the tracker will provide to that user a list with the peers that are offering the requested channel. These peers that are sharing the channel are named *hits*. Therefore, it becomes essential that the trackers have an updated hit list. NeuroCast has been implemented in such a way that all the users perform as trackers. Hence, any peer can share with another their hit list.

On the other hand, using a mesh approach forces us to introduce a new concept: *substreaming*. Substreaming appears as consequence of the need of receiving a stream from several sources at the same time. In this way, substreaming

consists in dividing the original stream in N parts, which are delivered from the different sources that are available following certain criteria that are explained later. Moreover, it could be interesting to receive more packets from one source than from another. Therefore, there are new issues to take care of, such as the arrival packet order or the packet distribution among the available sources. All the packets that the substream consists of are called *chunks*.

The actions that take place while joining the NeuroCast network are:

- *Getting the Hit List.* During the initialization, the NeuroCast application connects to the tracker, and it sends back a hit list of the requested channel.
- *Peer Selection.* Once a peer has the hit list, then it has to select the best set of peers from the list.
- *Packet Allocation.* NeuroCast sends the packets grouped in specific chunk numbers which are assigned to the peer transmitting the stream.
- *Network and load adaptation.* NeuroCast monitors the network status permanently while downloading the channel.

3.1 Entities

The number of processes in NeuroCast is variable and depends on the amount of established connections. As in Peercast, NeuroCast has a main thread which deals with the incoming requests and creates the children processes that handle these requests. In this section we briefly present the two main NeuroCast entities.

Servent. A servent handles requests from the system and sends the stream either to another NeuroCast instance or to the player. It acts as a server as well as a client. As in Peercast, there exists three different type of servents: direct, relay and server.

Channel. The channels are the main elements in NeuroCast. They are used to handle the different multimedia flows which are being shared in the network. Each flow uses a different channel. NeuroCast only distributes parts of the stream not the whole stream. Every channel has its original source. In general, this source is a user (*broadcaster*) who creates the channel and becomes the only physical link among the NeuroCast clients and the original multimedia file. The channel has two main elements.

- *The buffer*: It manages and stores the incoming packets temporally to allow a retransmission posterior. The maximum number of packets that the buffer can contain is 64.
- *The channel stream*: It deals with the handshaking and the channel reception. Channel stream class allows controlling the incoming packets.

NeuroCast also introduces the possibility of using several peers concurrently, dividing the stream among these, and making the network structure look like a grid. The subchannels are the application elements that allow to identify the stream fragments. Thus, each one of the sources will be associated to a subchannel.

3.2 Load balance

As mentioned in the previous section, NeuroCast allows to balance the load according to the network conditions. In this way, NeuroCast implements different algorithms to optimize the peer selection among the hits of the requested channel and the load distribution among the chosen ones. In the following we show how NeuroCast is able to adapt to the network conditions and redistribute the load according to these conditions.

Peers selection. One of the most critical parts in any player based on a P2P network is the selection of peers. Unlike conventional P2P applications for file distribution, the fact of working with video creates a hard and direct dependence on the peers that are transmitting the stream.

Once the hit list has been received from the tracker, it is necessary to calculate which is the subset of peers that will allow to download the stream with the best conditions regarding the network. NeuroCast will always try to prioritize those peers with a high available bandwidth, with the lower packet loss rate, and with a high availability. NeuroCast peer selection algorithm is based in CollectCast algorithm [5]. The strengths of this algorithm are the sources selection, the network status monitoring and the periodic source redistribution in order to adapt to the time-varying network conditions. The selection process can be split up into three phases:

1. Obtaining the hit list associated to the requested channel.
2. Enumerating the sets of hits that satisfy the constraints imposed by:

$$R_l \leq \sum_{P=\bar{P}_{act}} R_p \leq R_u, \tag{1}$$

   where Rp is is the maximum *sending rate* that a peer can contribute anytime during the session, Rl is the lower limit of the total sending rate of a set of peers, and Ru is the upper limit of the total sending rate of a set of peers.
3. Selecting the best set of peers among the solutions obtained previously.

When requesting a new channel, NeuroCast obtains a set of parameters from each source. Thus, the requesting peer obtains the `peerRp` of each source, i.e., the maximum *sending rate* that a peer can contribute with. Then, NeuroCast calculates the throughput and the losses of the link with each one the hits. In order to do these measurements, NeuroCast takes advantage of a modified version of the Iperf [8] application.

Apart from the bandwidth that a specific peer is willing to share, it is also interesting to know the availability of the set of hits. In any P2P network, it is impossible to determine the expected lifetime of the network, as it is not possible to predict when a node will be disconnected from the network, or the congestion of the network with the subsequent massive packet loss. A possible implementation to calculate the availability in our environment is carried out in the following way. A parallel process to NeuroCast runs to check the status of the hits during a limited period of time. With these measurements, it is generated a probabilistic function for each instant of the day, allowing to know the probability that a peer is connected at that time.

Packet Allocation. Unlike the CollectCast algorithm, in this first version of NeuroCast the final application is simplified and does not use FEC codes

to distribute the load among the different hits that have been selected to retransmit the channel.

The active peers ($\bar{P}_{act}$) collectively send the media file segment by segment: they all cooperate in sending the first segment, then the second one, and so on. The media file is divided into equal-length data segments. Peer p is assigned a number of packets D_p to send in proportion to its actual streaming rate:

$$D_p = \left\lceil \Delta \cdot \frac{R_p}{\sum_{x \in \bar{P}_{act}} R_x} \right\rceil . \tag{2}$$

where Δ is the size of original packets in which the media file is divided into. The D_p value is used to distribute the chunks among the different peers of the set. Delta is the number of chunks into which the stream is divided in order to work per groups with subchannels.

Network adaptation. To accommodate the maximum load of the network, NeuroCast adds a new functionality so that it is able to adapt at any moment the number of chunks that a peer is downloading. The variable used to determine the most appropriate chunk-distribution is the time between arrivals of packets. Thus, controlling the time between packet arrivals during the session, NeuroCast sets a threshold that allows to redistribute the number of chunks that each peer retransmits. This threshold is calculated following next equation:

$$threshold = \frac{\frac{buffer\_length}{bitrate} \cdot FACTOR\_SEP\_PACKETS \cdot numChunks}{numSubChannelChunks}. \tag{3}$$

As seen in equation 3, the threshold depends on different parameters:

1. It depends on the bitrate of the played video.
2. The `FACTOR_SEP_PACKETS` (maximum allowed delay between packet arrivals).
3. `numChunks` is related to the number of chunks into which the stream is divided among the different hits.
4. The `numSubChannelChunks` is the number of chunks that this subchannel retransmits.

To alert the rest of hits of the load redistribution the requesting peer sends to each one of the hits the message `CHANGE_PEER_CHUNKS` through it. Next, a NeuroCast server in the target machine reads the request and enables the flag associated with the redistribution, so that the `Relay` server that broadcasts the channel is ready to receive.

Once the subchannel is ready for the load redistribution, the peer performs the following steps:

1. Notifies the others subchannels about the redistribution.
2. Decreases by one the number of chunks that is downloading from its hit.
3. Searches for a subchannels with a lower `time_between_arrivals` in order to send the chunk it has you just subtracted.
4. Waits until the subchannel with lower `time_between_arrivals` has redistributed the chunks.

Once the rest of subchannels have been informed that a load redistribution is going to take place, they also enter into the process of redistribution. At

this point, two different situations must be distinguished: on one hand the subchannels with lower `time_between_arrivals` and on the other hand the rest of subchannels. The latter simply expect that the redistribution is done, to continue with the download of the new allocated chunks.

And regarding to the subchannel with the lowest `time_between_arrivals`, it follows these steps:

1. Checks that the channel has more than one subchannel currently. If that hit is the only active subchannel it must start to download the full stream from that hit.
2. If there are more subchannels, it increases by one the number of chunks of the subchannel.
3. Finally, it redistributes the load among all the hits from the list according to their number of chunks.

A different case from the one presented above, but that also requires load redistribution occurs when a peer does not receive packets from a certain subchannel. In these situations the load redistribution is forced immediately.

Another of the new features that have been introduced in NeuroCast is to get rid of the hits that are only sending a single chunk, and despite that, they still send packets with delay. In these cases, the peer removes that hit from its list of hits and downloads the latest chunk from another peer, leaving the subchannels on standby until the channel becomes unavailable or it decides to leave the network.

4 Performance Evaluation

In this section, we present emulation results by which NeuroCast capabilities are evaluated. Two different environments are configured in order to check NeuroCast's performance in heterogeneous network conditions. The reference scenario for our emulations is created using VNUML [9] as shown in Figure 1. In this virtual network, the $Host$ serves the video to $E0$, and the latter sends the stream to 3 machines ($E1$, $E2$ and $E3$).

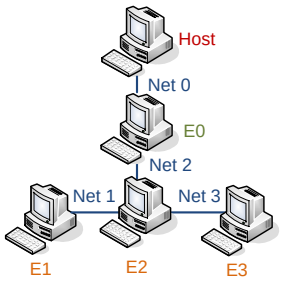


Fig. 1. Network topology used for the tests.

The main features of the virtual network and the virtual machines are shown in Table 1. Using this configuration, we change the network conditions where these peers are operating to recreate two different environments. These environments will emulate the conditions of a peer operating in a wireless network.

	Type	Bw
Net0	lan	2 Mbps
Net1	ppp	1 Mbps
Net2	ppp	1,5 Mbps
Net3	ppp	750 Kbps

	Host	E0	E1	E2	E4
maxRelays	1	4	4	4	4
peerRp	2 Mbps	1 Mbps	1 Mbps	700 Kbps	1 Mbps
minRp	100 Kbps	100 Kbps	100 Kbps	100 Kbps	100 Kbps
delta	0	10	10	10	15
factor_sep_pack	1	1.3	1.3	1.3	1.3
delay_margin	0	4	4	4	3
buffer_sep_pack	0	30	30	30	16
Rl	100 Kbps	100 Kbps	100 Kbps	100 Kbps	100 Kbps
Ru	2 Mbps	2 Mbps	3 Mbps	3 Mbps	3 Mbps

Table 1. Initial parameters of the virtual network and the virtual machines.

4.1 Impact of topology changes

In a wireless scenario, it is common to have highly changing topologies. In this section, we evaluate how NeuroCast achieves to adapt to these changing conditions without disrupting the quality of the multimedia service.

Thus, when downloading a video from more than one source, if one of these sources leaves the network, then the requesting peer redistributes the load and continues downloading the subchannels from other sources. The process that takes place consists in:

1. Checking if there is a free peer in the cached list of hits. That is, a hit from which the peer is neither downloading any chunk nor sending any part of the stream.
2. Requesting a new list of hits to the tracker and check again if there are any free hits.
3. Downloading the "missing" chunks from one of the hits that the peer is already using.

In the latter case it requires a reallocation of subchannels. Depending on the situation, NeuroCast carries out a series of changes:

- If the client is downloading more than one subchannel from the same hit, then it groups the chunks in the same subchannel.
- If a subchannel that is not at the end of the list is deleted, then this position is filled with the last subchannel, so there are no empty spaces in the list.
- If a subchannel downloads all the chunks into which the stream is divided, then the value of `numChunks` is set to 1 and the stream is no longer divided into different parts.

Next, we show an example of a peer leaving the network. Initially we have the following configuration (see Table 2):

Hit List	
Hit	**IP**
0 (tracker)	10.0.0.1
1	10.0.0.2
2	10.0.1.2
3	10.0.2.2

Subchannels List			
	IP address	**Number of Chunks**	**Chunks**
Subchannel 0	10.0.0.2	6	0 2 4 6 8 10
Subchannel 1	10.0.1.2	3	1 3 5
Subchannel 2	10.0.2.2	3	7 9 11

Table 2. Initial List of Hits and chunks distribution among the 3 subchannels.

After some time, source `10.0.0.2` leaves the network. The requesting peer checks that there are no-free hits in its hit list, so it requests a new hit list to the tracker (see Table 3).

Hit List	
Hit	**IP**
0 (tracker)	10.0.0.1
1	10.0.1.2
2	10.0.2.2
3	10.0.3.2

Subchannels List			
	IP address	**Number of Chunks**	**Chunks**
Subchannel 0	10.0.2.2	3	7 9 11
Subchannel 1	10.0.1.2	9	0 1 2 3 4 5 6 8 10

Table 3. New List of Hits and final chunks distribution among 2 subchannels.

As the only new addition to the list of hits is its own IP address, it has only the possibility of reusing the hits which were already transmitting the stream. In this example, as shown in Table 3 it is selected the `10.0.1.2` peer. Summarizing, NeuroCast is able to adapt to any change in the topology of the network no matter if the peer has left because it lost connectivity or because it decided to stop streaming.

4.2 Impact of traffic interferences

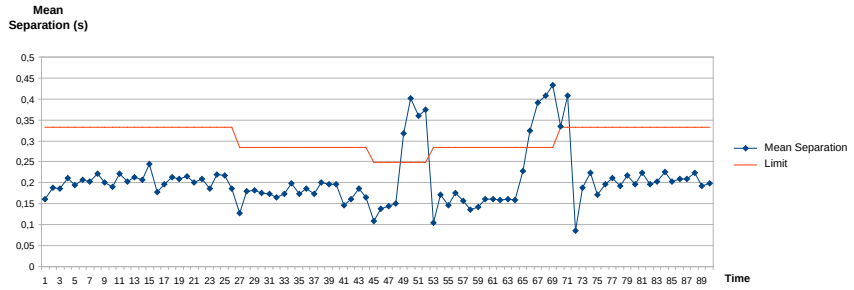
Another situation that is prone to happen in a wireless scenario is to suffer interferences from another peer in the network. To show how NeuroCast works under these conditions, continuing with the network topology of figure 1, we evaluate the situation where we have 4 machines running NeuroCast and playing a video: $Host$, $E0$, $E1$ and $E2$. As we have seen in previous cases, in a "stable" situation machine $E0$ downloads the stream from $Host$, machine $E0$ downloads it from $E1$, and finally $E2$ downloads it from $E0$ and $E1$ simultaneously.

In this section, we focus on studying the performance of machine $E2$ while suffering from traffic interferences. To achieve that environment, we introduce traffic interference in link $Net1$ using Distributed Internet Traffic Generator (D-ITG) [10] in $E3$.

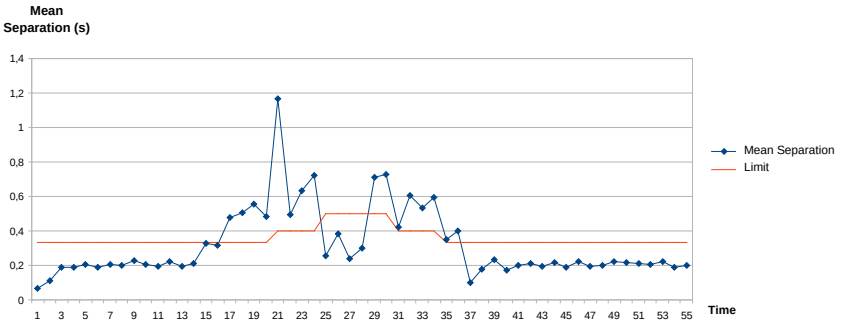
Prior to the injection the traffic interference, $E2$ is downloading 6 chunks from each source as the conditions of $Net1$ and $Net2$ are virtually identical. However, when we introduce a traffic of 800 kbytes/s in $Net1$ link, this balance is broken. Thus, chunks coming from the $E1$ are affected, and consequently, the time between packet arrivals is increased. Precisely this is reflected in the graphs of figure 2(a) and 2(b).

Figure 2 shows, apart from the mean time between packet arrivals, the theoretical limit that the program uses to determine when a redistribution of chunks is necessary. It is worth noting that the redistribution occurs when that limit is exceeded 3 times, because this is the value of the parameter `delay_margin` defined in the NeuroCast configuration file running in $E2$.

As we have seen previously, some parameters in the configuration file directly affect the behavior of the application during variations of the network conditions. In the following we carry out several setups, varying two parameters such as `delta` and `buffer_sep_packets`, and we observe in each case the time needed to redistribute the chunks.



(a) Subchannel 0.



(b) Subchannel 1.

Fig. 2. Mean time between packets arrival evolution.

To force the redistributions, we use Network Emulator (NetEm) [11] to change the link capacity of one of the hits that is retransmitting the stream. Thus we switch from the 1000 Kbps available at the beginning of the session, to 112 Kbps. Analyzing the results in Table 4 we observe that the parameter `Delta`, i.e. the number of chunks into which the stream is divided, does not affect significantly the obtained time, while the parameter `buffer_sep_packets` is decisive.

	Test1	**Test2**	**Test3**	**Test4**	**Test5**
Delta	20	5	20	20	5
`buffer_sep_packets`	20	20	50	5	5
Time to redistribution	90s	86s	173s	31s	31s

Table 4. Performance with a traffic interference of 90% of the link capacity.

Consequently, as it can be inferred from these results, any peer running NeuroCast even while suffering from interferences is able to operate without major quality of service degradation.

5 Conclusions

P2P technology gives novel opportunities to define an efficient multimedia streaming application but at the same time, it brings a set of technical chal-

lenges and issues due to its dynamic and heterogeneous nature. We must make a balance between the breadth and depth of a live streaming overlay tree. At the same time, the robustness issue must be considered carefully, as the dynamic feature and freedom of P2P network itself. In this this work we have presented the implementation of NeuroCast, an unstructured P2P system for video streaming that is able to adapt to wireless network conditions. NeuroCast design is based on an unstructured mesh-based architecture. It intends to optimize bandwidth allocation and combine dynamic peer selection strategies that rely on implicit feedback from data reception.

NeuroCast's performance is evaluated in emulated network environments. The experiments indicate that NeuroCast is capable of operating in a harsh environment taking into account network conditions. Our results also confirm the ability of unstructured mesh-based systems to withstand the high levels of transience that can result from user and network dynamics (churn, failures, congestion, etc.).

References

1. H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over a peer-to-peer network. Technical Report 2001-30, Stanford InfoLab, 2001.
2. Peercast: P2p broadcasting for everyone. [online] http://www.peercast.org, accessed on January 2011.
3. M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. *9th ACM Symposium on Operating Systems Principles*, 2003.
4. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for largescale peer-to-peer systems. *IFIP/ACM Intl. Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany*, 2001.
5. M. Hefeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev. Collectcast: A peer-to-peer service for media streaming. *ACM Multimedia*, 11:68–81, 2003.
6. Gnutella. [online] http://rfc-gnutella.sourceforge.net, accessed on January 2011.
7. X. Zhang, J. Liu, B. Li, and T. Yum. Coolstreaming/donet: A data-driven overlay network for efficient peer-to-peer live media streaming. *Proceedings of IEEE Infocom*, 2005.
8. A. Tirumala, M. Gates, F. Qin, J. Dugan, and J. Ferguson. Iperf: The (tcp/udp) bandwidth measurement tool, [online] http://iperf.sourceforge.net/, accessed on January 2011.
9. Virtual network user-mode-linux (vnuml). [online] http://www.dit.upm.es/vnuml, accessed on January 2011.
10. S. Avallone, S. Guadagno, D. Emma, A. Pescapè, and G. Ventre. D-itg distributed internet traffic generator. *International Conference on Quantitative Evaluation of Systems*, pages 316–317, 2004.
11. S. Hemminger. Network emulation with netem. In *Linux Conf Au*, [online] http://linux-net.osdl.org/index.php/Netem, accessed on January 2011.