GoalBit: The First Free and Open Source Peer-to-Peer Streaming Network

María Elisa Bertinat
L.P.E., Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Daniel De Vera
InCo., Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Darío Padula
L.P.E., Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Franco Robledo Amoza
L.P.E., Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Pablo Rodríguez-Bocca
InCo., Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
prbocca@fing.edu.uy

Pablo Romero
L.P.E., Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Gerardo Rubino
INRIA
35042 Rennes, France

ABSTRACT

In this paper we present the GoalBit platform. GoalBit is the first open source peer-to-peer system for the distribution of real-time video streams through the Internet. The main advantage of a P2P architecture is the possibility of exploiting the available bandwidth unused by the set of user' hosts connected to the network. The main difficulty is that these hosts are typically highly dynamic, they continuously enter and leave the network. To deal with this problem, we use a mesh connectivity approach (bittorrent-like) where the stream is decomposed into several pieces sent by different peers to each client. Nowadays, the GoalBit platform is used by operators and by final users to broadcast their live contents. In this paper we explain by the first time the Goalbit architecture, its protocol, and how the content is packetized. To illustrate its potential, we present some empirical results measured in a popular final user channel, with more than 60 peers concurrently connected.

Categories and Subject Descriptors

C.2.4 [**Computer-communication networks**]: Distributed systems

General Terms

algorithms, performance, measurement

Keywords

Streaming, P2P, QoE.

LANC'09 September 24-25, 2009. Pelotas, Brazil

1. INTRODUCTION

Video delivery applications are growing nowadays at increasing speed, as a consequence of the opening of video content producers to new business models, the larger bandwidth availability of the access network (on the Internet, on cellular networks, on private IP networks, etc.) and the explosion in the development of new hardware capable of reproducing and receiving video streams.

In the Internet context, video broadcast services are vastly deployed using Content Delivery Network (CDN) infrastructures, where a set of servers (located at strategic points around the Internet) cooperates transparently to deliver content to end users. However, since bandwidth is the most expensive resource on the Internet, and video delivery is one of the services that most demand it, live video services are limited yet in availability and diversity.

An increasingly popular alternative consists of using the often idle capacity of the clients to share the distribution of the video with the servers, through the present mature Peer to Peer (P2P) systems, which are virtual networks developed at the application level over the Internet infrastructure. The nodes in the network, called peers, offer their resources (bandwidth, processing power, storing capacity) to the other nodes, basically because they all share common interests (through the considered application). As a consequence, as the number of customers increases, the same happens with the global resources of the P2P network. Also, this approach helps in avoiding local network congestions, because the servers (other clients) can be extremely close of the final client.

1.1 P2P file-sharing systems

P2P networks are becoming more and more popular today (they already generate most of the traffic in the Internet). For instance, P2P systems are very used for file sharing and file distribution; some well known examples are BitTorrent [2], KaZaA [7], eMule [5], etc.

In BitTorrent, a list of the peers currently downloading each file is maintained by a central Web server (called the

tracker). Using this list, each peer knows, at any time, a subset of other peers that are downloading the same file (this subset is called the peer's swarm). The file is distributed in pieces (called chunks). Every peer knows which chunks are owned by the peers in its swarm, and explicitly pulls the chunks that it requires. Peers request the chunks which fewest number of peers have (*rarest-first* download policy), and peers upload chunks first to the peer requesters that are uploading data to them at the fastest rate (*tit-for-tat* upload policy).

This kind of protocols cannot be used directly for live video distribution, because the first chunks of the file are not downloaded first, and therefore the file cannot be played until the download has been completed. Also, these protocols were designed for the transference of one (or more) files of a fixed size; they don't support the transference of a continuous flow.

1.2 P2P live streaming

Live video P2P networks have to satisfy harder constraints, because video reproduction has real–time restrictions (file sharing don't) and the nodes only remain connected a few minutes [13]. Nowadays some commercial P2P networks for live video distribution are available. The most successful are PPlive [8] with more than 200,000 concurrent users on a single channel [11] (reaching an aggregate bit-rate of 100 Gbps), SopCast [12] with more than 100,000 concurrent users reported by its developers, CoolStreaming [3] with more than 25,000 concurrent users on a single channel [19], PPstream [9], TVAnts [14], and TVUnetwork [15].

PPLive [8] is the most popular P2PTV software, especially for Asian users. Born as a student project in the Huazhong University of Science and Technology of China, PPLive uses a proprietary protocol, and its source-code is not available. With reverse engineering, [6] and [1] show that it uses a bittorrent-like approach, with a channel selection bootstrap from a Web tracker, and peer download/upload video in chunks from/to multiple peers. It uses two kinds of video encoding formats, Window Media Video (VC-1) and Real Video (RMVB), with average bit rates from 250 Kbps to 400 Kbps, and it uses the user native media player to display the video. The distribution engine (BitTorrent-like protocol) does not use the *rarest-first* download policy, because the streaming must satisfy real–time restrictions. Also the *tit-for-tat* upload policy is not applied: when a client joins a channel, the other peers in the swarm send chunks forcefully to minimize the playback startup delay.

SopCast [12], created at the Fudan University of China, is very similar to the PPLive project. Following the same reverse engineering procedure, [6, 1] show that SopCast uses a bittorrent-like approach, with a proprietary protocol very close to the PPLive protocol. Read [11] for a complete performance study of this network.

CoolStreaming [3] (also known as DONet) is the predecessor of PPLive and SopCast, now forced to close down due to copyright issues. Also using a proprietary protocol, and closed source-code, as PPLive and SopCast, the CoolStreaming authors published information about its distribution engine [20, 18, 19]. The video stream is divided into chunks with equal size. A difference of CoolStreaming, with respect to PPLive and SopCast, is that the video stream is decomposed into six sub-streams by a rotating selection of video chunks.

In this paper we present the GoalBit platform. Its first difference with the previous mentioned commercial networks is that GoalBit is the first free and open source peer-to-peer platform for real-time video streaming in the Internet. Also, GoalBit is a complete suite for live media distribution, which implements the media generation, encapsulation, distribution and reproduction.

For the generation, the live media can be captured using different kind of devices (such as a capture card, a webcam, another http/mms/rtp streaming, a file, etc.), it can be encoded to many different formats (MPEG2/4-AVC, VC-1, ACC, VORBIS, MPGA, WMA, etc.) and muxed into various containers (MPEG-TS, ASF, OGG, MP4, etc). The streaming encapsulation, defined as GoalBit Packetized Stream (GBPS), takes the muxed stream and generates fixed size pieces (chunks), which are later distributed using the GoalBit Transport Protocol (GBTP). As we will explain later, the GBTP uses a bittorrent-like approach. Finally (at the client side) the GoalBit player reproduces the video streaming consuming the pieces obtained through GBTP. For the reproduction we integrate the Videolan Media Player [16] into our source code, therefore the final users does not need to install an external media player software.

Nowadays, GoalBit platform is used by operators (as ANTEL in their video portal AdinetTV[1]) and by final users to broadcast their live contents.

The paper is organized as follows. Section 2 presents the global architecture of the GoalBit platform. In Section 3 we introduce the GoalBit Transport Protocol. Section 4 presents the GoalBit Packetized Stream. In Section 5 we show the benefits of using GoalBit in real scenarios. Finally, the main contributions of this work are then summarized in Section 6.

2. THE GOALBIT PLATFORM

This section presents the network architecture, its components and their roles in the global structure.

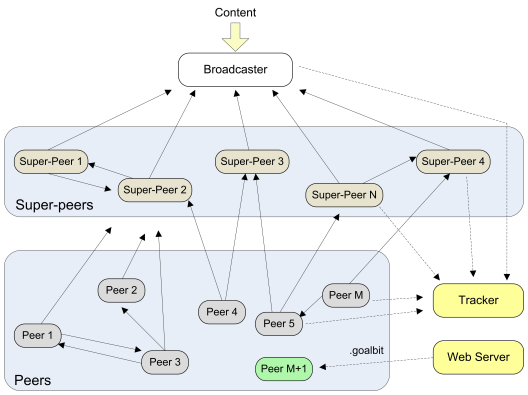


Figure 1: GoalBit architecture. The 4 components of the network and their connections are illustrated (the broadcaster, the super-peers, the peers and the tracker).

As Figure 1 illustrates, there are 4 different types of components in the network: the broadcaster, the super-peers, the peers, and the tracker. The broadcaster is the responsible for the content to be distributed, responsible for get-

[1] `http://www.adinettv.com.uy.`

ting it from some source and to put it into the platform. The super-peers are peers with a good capacity in terms of bandwidth, and their main role is to help in the initial distribution of the content. The peers are the final users, who connect themselves to some stream to play it. The tracker has the same role than in the BitTorrent network, in charge of the peers' management in the system. The information about the channels in the system is distributed by the Web. Usually, some Web servers host a base of `.goalbit` files (our channel configuration files, similar to the `.torrent` ones in BitTorrent), from which users get them.

In the sequel, we will describe with some detail each of these network components.

Broadcaster.

The broadcaster receives the content from some source (an analog signal, a streaming signal from the Internet, a stored video file, etc.), processes it, and puts it into the P2P network.

This critical component plays a main role in the streaming process. If it leaves the network, the streaming obviously ends. As a consequence, in a good design this component must not be overloaded by many tasks. Its main one is to create and transmit the content. It must then be avoided to concern it with an important participation in the distribution process itself.

Super-peers.

As previously stated, the broadcaster should not take care of the initial content distribution to the peers: this is the main task of the super-peers. These are highly available peers with a large uploading capacity. They get the streaming from the broadcaster and distribute it to the peers.

A first difference with the BitTorrent system is that in our architecture, the seeder concept depends on the type of peer. For the super-peers, the seeder is the broadcaster, while for peers, the seeders are the broadcaster (at which they don't normally have direct access) and all the super-peers (with which they communicate quite frequently).

Peers.

These are the final users of the system. They connect to a streaming, with the single purpose of playing it. They represent the large majority of nodes in the system (normally, they must be much larger in number than the super-peers). Peers exhibit a very variable behavior over time; in particular, they connect and disconnect frequently. Last but not least, they are the *raison d'être* of the network, the latter is built for them.

In future releases, GoalBit will promotes a peer to be a super-peer after it shows a good engagement with the network (at present, the peer type is defined at startup, and super-peers usually are hosts managed by the broadcaster).

Tracker.

The tracker maintains the list of peers (broadcaster, super-peers and peers) connected to every channel.

Each time some peer connects to a channel, it announces itself to the tracker. The latter registers the peer in a set associated with the channel, and returns a subset of other peers connected to the same channel. Presently, this is the only way to learn who is connected to each channel. It is possible to extend the GoalBit protocol to learn this subset also from the other peers connected to the channel (as the PEX or the Trackerless BitTorrent extensions).

The tracker has a critical supervising role in the GoalBit architecture. If it becomes out of service, the distribution becomes impossible at least for new connecting peers, that can not know who is able to feed them with pieces of content.

3. GOALBIT TRANSPORT PROTOCOL

As commented before the GoalBit Transport Protocol (GBTP) is an extended version of the BitTorrent Protocol, optimised for live video transmition. In this section we introduce some of its main characteristics and we present its whole specification.

3.1 Basic Concepts

In GBTP (as in Bittorrent) each piece of the stream is identified by an unique number, called the piece ID. It is used to accomplish the exchange of pieces between peers. As GBTP is designed to distribute a continuous streaming and not a file, the piece numbering is cyclic in the $[0, MAX\_PIECE\_ID]$ interval[2].

Each peer has a sliding window (over previous interval of possible ID values) associated with it. The sliding window has a minimum value (called the *base*) and a length (which defines the maximum window value). Peers only can download or share pieces whose ID are included in their respective windows

In order to reproduce the streaming, the peers have a player who consume in a sequential way the pieces stored in their sliding windows. The *execution index* is defined as the next piece ID to be consumed by the player.

Other important concepts used in the protocol are the *active buffer* and the *active buffer index* (or ABI). As it can be seen in Figure 2, the active buffer is defined as the consecutive sequence of pieces that a peer has, starting at the execution index. In other words, this corresponds to the downloaded video that the player will reproduce without interruption at a specific time. Finally, the active buffer index is the largest piece ID inside the active buffer.

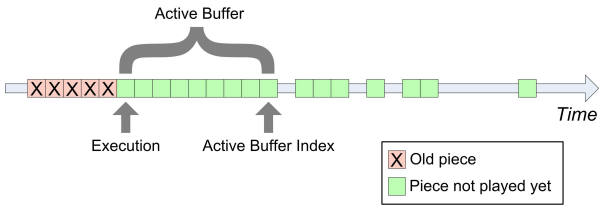


Figure 2: Active Buffer and Active Buffer Index (ABI)

Information stored in the Tracker.

As we mentioned before, a tracker manages multiple channels. For each channel the tracker stores a reference to the peers connected to it. Next we present in some detail the stored information associated with each peer.

Each channel has an identifier called *info_hash* (a 20-byte SHA1 string). Furthermore, each channel has information about the peers connected to him. For each peer, two different types of data are stored, called static data and dynamic

[2]Currently the MAX_PIECE_ID parameter is set to 2^{31}.

data. The static data is composed of the peer's IP, the port the peer is listening to for new connections, and the peer type, that is, one among the following: broadcaster, super-peer, peer, or broadcaster-super-peer[3]. Inside the dynamic data we can find for instance the peer's ABI, the last report date/time and the most recently measured (by the peer) quality of experience (QoE) value[4].

GoalBit files.

The GoalBit files (files with the `.goalbit` extension) are channel containers. Inside them there is the information needed to execute the streaming by a peer (i.e. tracker's address, channel's identifier, channel's metadata, etc.). These files are very similar to the corresponding ones in the BitTorrent protocol (with the `.torrent` file extension), except that GoalBit files are plain text files structured using XML. For each channel the data stored is presented in Table 1. As a GoalBit file could contain information about multiple channels, it is possible to set a default channel, to be executed when the file is opened.

Table 1: Channel information stored in a GoalBit file.

Parameter	Description
channel_id	Channel identifier (a string of chars).
chunk_size	Piece size for the execution.
tracker_url	Tracker URL where the channel is executed.
bitrate	Streaming bitrate.
name	Channel name.
description	Channel description.
thumb	Channel logo.

3.2 Tracker-peer communication

There are two types of exchanges between tracker and peers: the initial ones, and the exchanges that take place periodically during the sojourn of the peer in the network. These communications are carried out over the `HTTP/HTTPS` protocol and are based on a unique message called *announce*. In Table 2 we show the parameters sent by the peer in the announce request to the tracker and in Table 3 those sent by the tracker in its answer to the peer's request.

Initially, the peers contact the tracker in order to get a list of peers and an index from where to start asking for pieces. Normally, the peer sends an announce to the tracker with $numwant = 55$ and $ABI = MAX\_PIECE\_ID + 1$. In the answer, the peer gets a list of peers with at most *numwant* elements, the starting index and the peer type assigned.

Since the tracker needs to store status information for each peer, there must be a regular communication between them to allow this transfer of information. In the periodic communication (and leaving aside some particular cases), the peer sends an announce with $numwant = 0$, $ABI =$ the current ABI, and $QoE =$ the quality of experience measured by the peer in the last period. In this case, there is no relevant information in the tracker's answer. At those exchanges, peers can also ask for more peers. These exchanges are regulated by the tracker, who defines the minimal time interval between two consecutives announces.

[3]The broadcaster-super-peer type is used in the networks where there are no super-peers. This is a simplified version of the GoalBit architecture, where a broadcaster serves directly to the peers.

[4]The QoE is measured using the PSQA (Pseudo-Subjective Quality Assessment) methodology described in [10].

Table 2: Parameters sent by the peer in the announce request to the tracker.

Parameter	Description
protocol	Protocol version used, currently "GoalBit-0.4".
info_hash	Channel identifier (20-byte SHA1 string).
peer_id	Peer identifier. This is a 20 bytes random string, generated by the peer at the beginning of the session.
event	This parameter must be "started" or "stopped". During the initial and the periodical communication it must be "'started". The "stopped" value must be sent only in the peer disconnection.
port	Port where the peer listens to incoming connections.
uploaded	Uploaded bytes since the beginning of the execution.
downloaded	Downloaded bytes since the beginning of the execution.
numwant	Number of peers that the tracker should return to the peer. If it is 0, the tracker doesn't return any peer.
ABI	Current peer's ABI (while sending this message). In the case of the initial communication (when the peer doesn't still have an ABI), the value *MAX_PIECE_ID + 1* must be sent.
peer_type	Peer type: Broadcaster = 1, Super-peer = 2, Peer = 3, Broadcaster-super-peer = 4.
peer_subtype	Peer subtype (only used in particular cases).
QoE	Quality of experience measured by the peer.
Compact	If this parameter is set to 1, that means that the peer accepts a compact answer (in the tracker response each peer will be sent as a 6 bytes string where the first 4 bytes represent the peer IP and the last 2 bytes the peer port).
tracker_id	Inside the announce response the tracker can include a parameter called *tracker_id*, if this happened, this should be sent back to it, in the next announces.

For the two types of announcements (initial and periodical), the tracker sends different list of peers (the swarms) depending on the type of peer: for the broadcasters, there is no swarm to deliver; the super-peers must receive the list of the broadcasters and of the other super-peers; the peers must get a list with a few super-peers and many peers. There are many possible strategies to build a swarm by a tracker. Nowadays, there is a very popular one, called P4P [4, 17], that suggests that the swarm to be sent to a peer must be composed of peers geographically close to the one that is initiating its connection to the network. Moreover, it is desirable that the peers in the swarm are downloading chunks temporally close to those needed by the peer that will receive the list (that is, chunks close to the ABI of the receiver).

3.3 Peer-to-peer communication

As we have previously seen, the tracker sends to every new peer in the network a subset of other peers watching the same channel and an index in the flow of chunks from which to start to view. Once with this information, the exchanges between peers can start; they are performed by exchanging messages using the TCP protocol.

Two kind of messages are exchanged. The first ones are used to inform about the contextual situation of each peer, i.e. which peer remains connected and which pieces already have been downloaded by every peer. To exchange contextual information five type of messages are used: HAND-

Table 3: Parameters sent by the tracker in its answer to the peer's request.

Parameter	Description
interval	Time interval (in seconds) that the peer should wait before sending a new announce to the tracker.
tracker_id	This parameter should be send back to the tracker in the next announces.
broadcaster_num	Broadcaster-peers number involved in the streaming.
super-peer_num	Super-peer number in the streaming.
peer_num	Peer number in the streaming.
max_ABI	Largest ABI reported by a seeder.
peer_type	Peer type assigned to the announcing peer.
offset	Starting index (where the peer should start asking pieces). This parameter is only used in the initial communication.
peers	Peers list.

SHAKE, BITFIELD, HAVE, WINDOW UPDATE and KEEP-ALIVE. The other group of messages are used to exchange pieces. As we will show later, the process to exchange a piece is non trivial, and eight types of messages are used: INTERESTED, NOT INTERESTED, CHOKE, UNCHOKE, REQUEST, CANCEL, PIECE and DONT HAVE. We present now with some detail the messages defined in the protocol.

Exchanging contextual information.

Initially, the peer sends messages of the type HANDSHAKE (see the details in Table 4) to each peer present in the list that the tracker sent to it, in order to establish a connection with them. Among other things, in that message is sent information such as the type of peer and the sliding window used by it (the base and the width). This data is used by the other peers to accept or reject the connection with the new one. In case of accepting it, the receiver must send back another HANDSHAKE message with their own data.

Table 4: Definition of the "Handshake" message. This is the initial message in the communication between 2 peers. Technically, it must be sent from each one of them to the other when a new connexion is established. The message is 77 bytes length.

Field	Size (bytes)	Description	Value
pstrlen	1	Here the length of the field *pstr* is given	16
pstr	*pstrlen*	Protocol identifier (a string of chars)	GoalBit protocol
Reserved	8	Byte reserved by the protocol	0
info_hash	20	Identifier of the transmision. This field is a hash SHA1 of 20 bytes of the identifier of the concerned channel	N/A
peer_id	20	Identifier of the peer. A random string of 20 bytes for the user's identification	N/A
peer_type	4	Type of peer: broadcaster peer = 1, super-peer = 2, peer = 3, broadcaster super-peer = 4	N/A N/A
offset	4	Base of peer's sliding widow	N/A
length	4	Length of peer's sliding widow	N/A

Once the communication between two peers is established, they will exchange messages of the type BITFIELD (see the details in Table 5), in which the ID of the pieces that each one has in the window of the other one are sent (using the data about the sliding windows sent in the HANDSHAKE messages). In this way, a given peer keeps informed about the chunks the rest of the peers have inside its own window (remember that the protocol forces each peer to receive chunks only inside its window).

Table 5: Definition of the "Bitfield" message. This message informs about the pieces that a peer has in a specific window. It has a variable length depending on the size of specific window. It is sent immediately after the HANDSHAKE process or as an answer for the WINDOW UPDATE message. The seeders send a null Bitfield (*len*=0) in these messages.

Field	Size (bytes)	Description	Value
len	1	Message length	$5 + X$
id	1	Message identifier	5
offset	4	Base of the bitfield	N/A
bitfield	X	Bits sequence, where each bit announces if the peer has a piece or not. Each piece is identified respect to the offset (the Nth bit represents the piece with $ID = offset + N$)	N/A

To have this information up to date, each time a peer A receives a new chunk it must send a message of the type HAVE (see the details in Table 6) to any other connected peer B that satisfy the following conditions:

- The received chunk belongs to B's window.
- Peer B does not have the piece at his disposal (a peer will never ask for a chunk it already has, so, it makes no sense to send it a HAVE message in that case).
- Peer B's ABI must be smaller than the ID of the chunk (a peer will never ask for a chunk whose ID is less than his ABI, no interest for it in doing that).

Table 6: Definition of the "Have" message. This message acknowledges the reception of a new chunk. It's length is fixed, and equal to 10 bytes.

Field	Size (bytes)	Description	Value
len	1	Message length	9
id	1	Message identifier	4
piece_id	4	Identifier of the chunk received	N/A
ABI	4	Value of the current ABI of the peer	N/A

It also happens that peers move forward their sliding window getting a new base. In these cases, the change must be immediately communicated to the set of connected peers by means of a message of the type WINDOW UPDATE (see message details in Table 7). Peers that receive the message must answer with a BITFIELD message providing the information about the chunks that they already have in the new window of the original peer

Peers must periodically exchange messages between them. If after a specified delay (normally about 2 minutes) there has been no exchange between two given peers, they must exchange a KEEP-ALIVE message (see message definition

Table 7: Definition of the "Window Update" message. This message is sent by a peer that changes its sliding window. It has a fixed length of 6 bytes. It has to be answered with a BITFIELD message.

Field	Size (bytes)	Description	Value
len	1	Message length	5
id	1	Message identifier	11
offset	4	New base of the bitfield	N/A

in Table 8); on the contrary, the connexion will be assumed to be closed.

Table 8: Definition of the "Keep-Alive" message. Message used to keep active a connexion. It has a fixed length equal to 1 byte.

Field	Size (bytes)	Description	Value
len	1	Message length	0

Exchanging pieces.

We focus now on the way chunks are exchanged. So far, we discussed about the way peers connect and exchange information about their respective contexts (i.e. which pieces already have been downloaded by every peer). With this information, they must decide if they are interested in downloading from any specific peer. If they are, they must send a message of the type INTERESTED (see message details in Table 9). If they have been in such an exchange and if that is not anymore the case, the appropriate message to send is of the type NOT INTERESTED (see message details in Table 10).

Table 9: Definition of the "Interested" message. This message is used by peers in order to show their interest in downloading data from other peer. It's length is fixed, equal to 2 bytes.

Field	Size (bytes)	Description	Value
len	1	Message length	1
id	1	Message identifier	2

The protocol specifies that for a peer A to download a chunk from peer B, it is necessary that A is interested in exchanging with B, and that B allows A to download. Peers accept and reject other peers to download from them by means of the CHOKE and UNCHOKE messages (see messages definitions in Table 11 and Table 12 respectively).

When a peer is authorized to download chunks from another one (that is, it sent an INTERESTED and received an UNCHOKE), the latter could ask for parts of a chunk (called *slices*; chunks are always exchanged in slices, as in BitTorrent) by means of a REQUEST message (see more details in Table 13).

It is always possible, for a peer, to cancel the request for a slice through the message CANCEL. Finally, the peer having received a REQUEST sends the slice using a PIECE message (see CANCEL and PIECE messages definitions in Table 14 and Table 15 respectively).

Table 10: Definition of the "Not Interested" message. This message is used by peers in order to state that they do not have any interest in downloading data from some peer. It's length is fixed, equal to 2 bytes. It is only sent when an INTERESTED message has been previously sent (from the same sender to the same receiver).

Field	Size (bytes)	Description	Value
len	1	Message length	1
id	1	Message identifier	3

Table 11: Definition of the "Choke" message. If peer *A* sends this message to peer *B*, the latter can not download from the former. It has a fixed length of 2 bytes.

Field	Size (bytes)	Description	Value
len	1	Message length	1
id	1	Message identifier	0

Table 12: Definition of the "UnChoke" message. If peer *A* sends this message to peer *B*, the latter is allowed to download from the former. It has a fixed length of 2 bytes.

Field	Size (bytes)	Description	Value
len	1	Message length	1
id	1	Message identifier	1

Table 13: Definition of the "Request" message. This message is used by peers in order to request the download of a slice of a piece. It's length is fixed, equal to 14 bytes.

Field	Size (bytes)	Description	Value
len	1	Message length	13
id	1	Message identifier	6
piece_id	4	Requested piece identifier	N/A
begin	4	Begin of the slice (in bytes respect to the piece length)	N/A
length	4	Slice length (in bytes)	N/A

Table 14: Definition of the "Cancel" message. This message is used by peers in order to cancel the request of a slice of a piece. It's length is fixed, equal to 14 bytes.

Field	Size (bytes)	Description	Value
len	1	Message length	13
id	1	Message identifier	8
piece_id	4	Canceled piece identifier	N/A
begin	4	Begin of the slice (in bytes respect to the piece length)	N/A
length	4	Slice length (in bytes)	N/A

Table 15: Definition of the "Piece" message. This message sends a slice of a piece. It has a variable length depending on the slice length. It is the answer for the REQUEST message.

Field	Size (bytes)	Description	Value
len	1	Message length	$9 + X$
id	1	Message identifier	7
piece_id	4	Piece Identifier	N/A
begin	4	Begin of the slice (in bytes respect to the piece length)	N/A
block	X	Data of the slice	N/A

If a peer is faster than its seeder (a super-peer), it is possible that the peer requests a piece that has not been generated yet in the seeder. In this situation a DONT HAVE message is answered by the seeder. It is a kind of bitrate control inline the protocol. It could also happen between super-peers and the broadcaster. See the DONT HAVE message definition in Table 16.

Table 16: Definition of the "Don't Have" message. This message is used by seeders in order to answer requests about pieces not yet generated. It's length is fixed, equal to 6 bytes.

Field	Size (bytes)	Description	Value
len	1	Message length	5
id	1	Message identifier	10
piece_id	4	Piece Identifier	N/A

3.4 Strategies applied in the protocol

We describe here different strategies that can be applied in the protocol. The first one concerns the selection of peers for the piece exchanging, and the second one the selection of pieces to download.

Peer selection: the strategy to allow a peer to make downloads.

The strategy used for choosing which peers are going to be enabled to download data is the same than the one used in BitTorrent, namely the *tit-for-tat* one. It consists in the following principle: a peer must enable those peers from which it donwloads the largest number of chunks, a sort of award to the most generous connected peers (those who share the most with the rest of the community). To this is added an "optimistic" policy to explore the network and to integrate arriving peers, called *optimistic-unchocking*. It consists in choosing a peer at random, without taking into account how generous it has been in the past. More in detail, the pseudo-code of the peer selection algorithm is:

- Time is divided into cycles of T-seconds each, which are numbered.
- At the end of cycle i, N peers are selected to be enabled. If iT is a multiple of M, then *tit-for-tat* is applied to select the best $N-1$ peers and the other peer is selected by random applying *optimistic-unchocking*. If iT is not a multiple of M, then *tit-for-tat* is applied to select N peers[5].

Piece selection: the strategy to define which piece will be requested.

Another key strategy concerns the chunk selection process, that is, the main decision to take when a new chunk must be requested to some peer: which chunk to ask for. In BitTorrent, the strategy used is called *rarest-first*: peers must select those chunks that are the less present in the system (the rarest ones). The goal is to reach a distribution of chunks in the network as uniform as possible, thus reducing the chances that a peer can not complete the downloaded file. In our case of real-time video streaming, it has been already observed that this strategy leads to large initial buffering, which is not a desirable behavior. Here, a tradeoff between the initial buffering and the continuity of the playing process must be found. It is simple to show that a *greedy* strategy in which peers always request the next missing chunk from the last played one provides very good values for the initial buffering phase but poor continuity in video playing [21]. For this reason, in GoalBit we follow a hybrid approach between these two methods (*rarest-first* and *greedy*).

As it can be seen in Figure 3, we define three buffer ranges with respect to the execution index, called "urgent", "usual" and "future", and apply the following algorithm:

- If a chunk is missing in the "urgent" range, that chunk is requested; if more than one are missing, the first to be played is requested.
- If all chunks in the urgent range have been downloaded, the chunk selection is done based on the result of sampling from an exponential random variable, which gives higher weight to the "usual" range, but allowing to ask for chunks in the "future" range, from time to time.

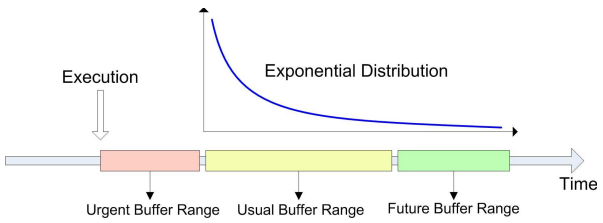


Figure 3: Chunk selection strategy.

4. GOALBIT PACKETIZED STREAM

The previous section presents the GBTP protocol. GBTP specifies how the pieces of live content should be distributed, but does not specify how the pieces should be generated and encapsulated. GoalBit Packetized Stream (GBPS) specifies the content and structure of each one of the pieces to be streamed through GBTP.

Usually, the following steps are involved in the live content generation and codification:

[5]Usual values are: $T = 10$ sec, $N = 4$, $M = 3$. In words, the set of enabled peers is re-defined every 10 sec, and every 30 sec an optimistic enabling is performed.

1. Content capture: the content (audio and video) is captured from an analogic signal.

2. Analogic signal codification: in this step the audio and video signals are separately coded to some known specification (e.g.: MPEG2/4-AVC, VC-1, ACC, VORBIS, MPGA, WMA, etc.). As result at least 2 elementary streams are generated (one for the video and one for the audio).

3. Elementary streams multiplexing and synchronization: this step consists in the generation of a single stream of bytes that contains all of the audio, video, and other information needed in order to decode them simultaneously. Some well known specifications (called muxers) are MPEG-TS, MPEG-PS, MP4, ASF, etc.

4. Content distribution: the single stream generated by the muxer is distributed using some transport protocol (RTP, HTTP, UDP, RTMP, GBTP, etc.).

The GBPS specification is located between the third and the fourth steps (after the content was multiplexed and before the content is distributed). The GBPS packets have a fixed length (defined by the broadcaster at its start up). The payload of a GBPS packet is a sequence of muxer packets (packets generated by the muxer). Muxer packets could have variable size (depending of the multiplexing specification). Therefore, in order to fulfill the GBPS pieces with muxer packets is necessary to fragment some muxer packets. Figure 4 shows the GBPS piece structure, the fields *i_data_start* and *i_data_end* are used for supporting the muxer packets fragmentation.

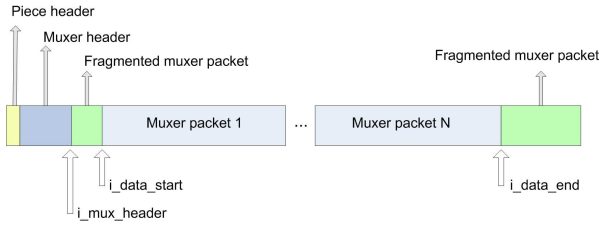


Figure 4: GBPS piece structure.

An important feature of the GBPS specification is the fact that it was designed to be independent of the chosen muxer, what gives the possibility of distributing any type of streaming.

4.1 Different muxer specifications

All multiplexing specifications have a common purpose, to multiplex and synchronize a set of elementary streams, but between them large differences can be found. Some of these were designed for live content distribution (such as MPEG-TS) and some others were designed for a video on demand scenario (such as MPEG-PS, ASF, etc.). This does not mean that is imposible to stream live content using ASF or MPEG-PS: the point is that when we use these specifications, we need to add extra data into the stream in order to decode it correctly.

Specifications designed to live content distribution, periodically insert multiplexing information (i.e.: number of elementary streams multiplexed, each elementary stream id, etc.) and synchronization data inside the stream. The other specifications store this information in a special packet called the muxer header, which must be the first packet delivered to the client (because without the muxer header it is not possible to decode the stream).

In order to be a generic specification and support any muxer, GBPS must consider the existence or not of a muxer header. All GBPS pieces have their own header. In Table 17 the format and content of this header are shown. In case of muxer specifications requiring the delivery of a muxer header, the latter is periodically added to the GBPS pieces using the field *i_mux_header* (usually, a muxer header is inserted every 10 GBPS pieces). Given that the muxer header could change in the middle of an execution, there is the possibility of announcing it to the whole community through the field *i_flag*. If the first bit of this field is set to 1, the muxer header should be sent again to the player.

Table 17: GBPS Header format.

Position (bytes)	Field	Description
0..3	*i_data_start*	Start of the first non-fragmented muxer packet.
4..7	*i_data_end*	End of the last non-fragmented muxer packet.
8..11	*i_mux_header*	Muxer header size (0 if it doesn't exist)
12	*i_flags*	Protocol flags:

Position (bits)	Description
0	1 means that the muxer header has changed.
1..7	Reserved (not in use)

5. EMPIRICAL RESULTS

The GoalBit platform is being used by some content distribution networks and by final users in order to distribute their live content over the Internet. In this section we present some measures of a typical popular final user channel.

The broadcaster infrastructure is a single host with an outgoing bandwidth capacity between 10 and 20 Mbps. This host executes 6 broadcaster-peers and 6 super-peers. The stream is encoded in H.264/ACC having a bitrate of 300 Kbps, and it is multiplexed in ASF. The GBTP piece length is 65536 bytes.

The test period is 2000 seconds (more than half an hour). Most of the results are measured at the client side, by parsing and analyzing logs generated by a client (our client) during the streaming period. At the server side, only the outgoing and incoming bandwidth rates are measured.

Evolution of the number of peers in the network.

Figure 5 shows the evolution of the number of peers connected to the network during the streaming period. There are 6 broadcaster-peers, 6 super-peers (all these belonging to the streaming owner) and an average of 62.3 peers. This information can be measured at the client side because of the GBTP design, in which the tracker always reports the total number of peers connected to the stream.

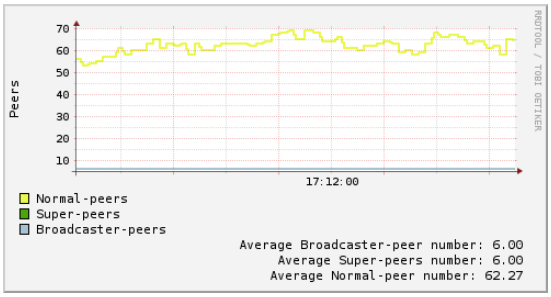
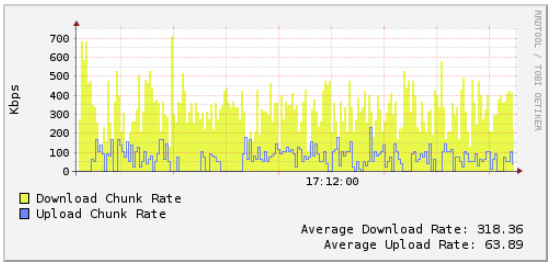
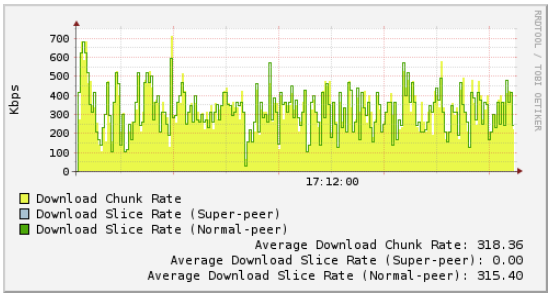


Figure 5: Evolution of the number of peers in the network.

Download bitrate vs upload bitrate.



(a) Download bitrate vs upload bitrate.



(b) Download piece rate (in Kbps) discriminated by peer type.

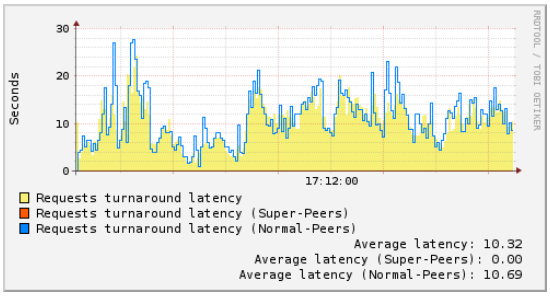
Figure 6: Empirical measured bitrates.

Figure 6(a) shows the evolution of the download bitrate and the evolution of the upload bitrate (both measured in Kbps). Our peer downloads and uploads pieces at an average bitrate of 318 Kbps and 64 Kbps respectively (it uploads the 20% of what it downloads). This difference could be explained by our connectivity restrictions (where the maximun uploading bandwidth is set to 100 Kbps).

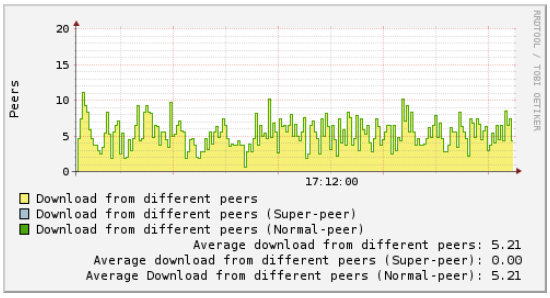
In Figure 6(b) we present the download piece rate (in Kbps) given by peer type. Our peer downloads the 100% of the stream from other peers (note that in the period we never downloaded a piece directly from a super-peer).

Latency vs downloads from concurrent peers.

We define latency as the time period between the request of a piece and its final reception. Figure 7(a) presents the latency (in seconds) by peer type. We observe an average latency of about 10 seconds. Given that the streaming was encoded at a bitrate of 300 Kbps and the GBTP piece size was set to 65536 bytes, the video player should consume 1 piece every 1.7 seconds. Therefore, if we have a latency of 10 seconds, in order to satisfy the piece consuming rate our peer should perform parallel requests to different peers. In Figure 7(b) we present the number of concurrent downloads from different peers, aggregated in periods of 10 seconds. It shows that every 10 seconds, a peer performs 5.2 piece downloads from different source peers, thus balancing the piece consuming rate.



(a) Latency (in seconds) per peer type.



(b) Concurrent downloads from different peers grouped periods of 10 seconds.

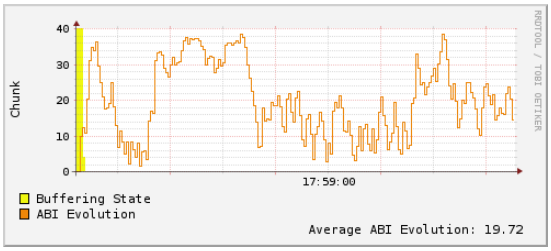
Figure 7: Empirical measured latency and concurrence.

ABI evolution and piece losses.

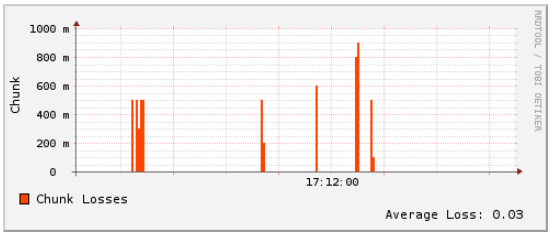
Figure 8(a) shows the average number of pieces inside the active buffer (i.e., the downloaded video that the player will reproduce without interruption). On the other hand, Figure 8(b) presents the piece losses occurred during the streaming period (i.e.: the pieces needed by the player that were not downloaded in time). As the reader can see, both graphics are correlated: every time the number of pieces inside the active buffer is close to zero, piece losses are observed. These figures also show that the average number of pieces in the active buffer is close to 20 pieces. This value is higher than the initial buffer size, set to 16 pieces, and therefore the average piece losses is very small (measured as less than 0.03%).

Bandwidth consumed in the server.

Figure 9 shows the bandwidth consumed by the server, over time. It shows 2 main peaks in the outgoing bandwidth rate, corresponding to the streaming of 2 different popular events. All the results shown previously are focused on the second one, where the mean rate is around 2 Mbps.



(a) Number of pieces inside the active buffer.



(b) Piece losses occurred during the streaming period.

Figure 8: Empirical measured buffer.

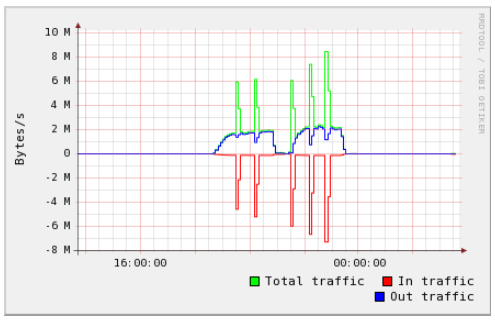


Figure 9: Bandwidth consumed in the server during the streaming period.

6. CONCLUSIONS

This paper presents GoalBit, a complete platform for live video distribution. The architecture of the Goalbit network and the protocol specification are described.

By means of the first empirical results obtained by means of the observation of a streaming with more than 60 peers concurrently connected, we present some of the potentials of Goalbit: inexpensive streaming (in terms of bandwidth consuming) and high quality of experience for final users. Comparing the GoalBit platform with a classical client-server streaming infrastructure, in the tested period, we could measure savings of almost 90% in bandwidth consumption: specifically, in order to distribute a 300 Kbps classical streaming to 62 users, we need around 18 Mbps of upload bitrate, instead of the 2 Mbps of bandwidth used by GoalBit. On the other hand, our client only measured a 0.03% of losses during the streaming period.

Another important feature of our platform is that it is the first free and open source peer-to-peer streaming network available in Internet.

7. REFERENCES

[1] S. Ali, A. Mathur, and H. Zhang. Measurement of commercial peer-to-peer live video streaming. In *In Proc. of ICST Workshop on Recent Advances in Peer-to-Peer Streaming*, Weaterloo, Canadda, 2006.

[2] Bittorrent home page. http://www.bittorrent.org, 2007.

[3] CoolStreaming home page. http://www.coolstreaming.us, 2007.

[4] DCIA Activities. P4P Working Group (P4PWG). http://www.dcia.info/activities/#P4P, 2009.

[5] eMule home page. http://www.emule-project.net, 2007.

[6] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. Insights into pplive: A measurement study of a large-scale p2p iptv system. In *In Proc. of IPTV Workshop, International World Wide Web Conference*, 2006.

[7] KaZaA home page. http://www.kazaa.com, 2007.

[8] PPLive Home page. http://www.pplive.com, 2007.

[9] PPStream home page. http://www.ppstream.com/, 2007.

[10] P. Rodríguez-Bocca. *Quality-centric design of Peer-to-Peer systems for live-video broadcasting*. PhD thesis, INRIA/IRISA, Université de Rennes I, Rennes, France, april 2008.

[11] A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, and S. Tewari. Will iptv ride the peer-to-peer stream? *Communications Magazine, IEEE*, 45:86–92, 2007.

[12] SopCast - Free P2P internet TV. http://www.sopcast.org, 2007.

[13] K. Sripanidkulchai, B. Maggs, and H. Zhang. An analysis of live streaming workloads on the internet. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 41–54, New York, NY, USA, 2004. ACM Press.

[14] TVAnts home page. http://cache.tvants.com/, 2007.

[15] TVUnetworks home page. http://tvunetworks.com/, 2007.

[16] VideoLan home page. http://www.videolan.org, 2007.

[17] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz. P4P: Portal for (P2P) Applications. In *Proceeding of the conference of the Special Interest Group on Data Communications, a special interest group of the Association for Computing Machinery (SIGCOMM'08)*, Seattle, USA, August 2008.

[18] S. Xie, G. Y. Keung, and B. Li. A measurement of a large-scale peer-to-peer live video streaming system. In *ICPPW '07: Proceedings of the 2007 International Conference on Parallel Processing Workshops (ICPPW 2007)*, page 57, Washington, DC, USA, 2007. IEEE Computer Society.

[19] X. Zhang, J. Liu, and B. Li. On large scale peer-to-peer live video distribution: Coolstreaming and its preliminary experimental results. In *IEEE International Workshop on Multimedia Signal Processing (MMSP'05)*, Washington, DC, USA, 2005. IEEE Computer Society.

[20] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming. In *IEEE Conference on Computer Communications (INFOCOM'05)*, Washington, DC, USA, 2005. IEEE Computer Society.

[21] Y. Zhou, D. M. Chiu, and J. Lui. A Simple Model for Analyzing P2P Streaming Protocols. In *Proceeding of the IEEE International Conference on Network Protocols (ICNP'07)*, pages 226–235, Beijing, China, October 2007.