TREBALL DE FI DE CARRERA

Títol: PPSP: Analysis, design and proof of concept

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Jaume Borràs Fuentes

DIRECTOR: Jesús Alcober Segura
Co-Director: Alberto José González Cela

DATA: 20 Juliol 2010
Resum

En aquest projecte hem dissenyat l’arquitectura utilitzada en els missatges d’un protocol, aquest utilitza com a referència els requeriments especificats a la primera versió de l’especificació del protocol PPSP. Hem extret la base del protocol del IETF Working Group que treballa actualment en el mateix camp.

PPSP és un protocol centrat principalment en el vídeo streaming i que te com a objectiu l’intercanvi de media entre diferents Peers. Ens hem centrat en la manera que els Peers anuncienc a altres Peers quins chunks tenen i quins els hi manquen. També hem desenvolupat la manera en que els grups de Peers (que formaran Partner Lists) intercanvien informació amb els seus Trackers, encarregats dels canals de vídeo streaming.

La finalitat d’aquest projecte és crear un protocol basat en el PPSP WG, fent possible que una única aplicació pugui reproduir el contingut streaming provinent de canals de televisió de diferents companyies.

L’estandardització del protocol s’aplicarà en diferents punts del flux i en nodes diferents que ofereixen serveis diferents. Ens hem centrat especialment en les funcionalitats de senyalització dels Peers i del Tracker.

El protocol serà efectiu a nivell d’usuari també. Totes les aplicacions de streaming P2P, sense importar de quin nivell ni finalitat, podran treballar amb el protocol estandarditzat i ja no serà necessari l’ús de protocols privats. Actualment cada canal/companyia utilitza el seu propi protocol, el que condiciona que sigui necessari un reproductor diferent per a cada companyia.

Un altre aspecte atractiu del nou protocol es que oferirà la possibilitat de fer servir els mateixos equips per treballar com a Tracker i Bootstrap Server i que a l’hora podran ajudar en la distribució del flux en llargues distàncies. Avui en dia, les empreses utilitzen aplicacions privades que realitzen les funcions descrites prèviament. Estandaritzant el protocol, totes les empreses que ho desitgin podran compartir els equips, facilitant l’aparició de noves empreses i de nous canals.

Utilitzant tècniques de Source Codinig, adaptarem la qualitat del flux d’imatge a les capacitats de cada receptor.

També implementarem funcionalitats pròpies del protocol ALTO, que ens permetran ajustar els criteris de selecció de paretles (Peer-Partners) a les nostres necessitats i qualitats.

El programa CoolRuc, desenvolupat per I2CAT, ha servit com a programa de referència de P2P Streaming en aquest projecte.
Summary

In this project we have designed the message architecture for a protocol using as reference the first requirements specification of PPSP. The bases of our protocol have been obtained from the IETF working group who is currently working on this same topic.

PPSP is a protocol designed to exchange media between peers, focused on video streaming. We have centered our project on the way peers will announce to other peers which chunks it has and which are missing. How peer groups (partner lists) are formed and how peers exchange information with the trackers in charge of a channel flows.

The aim of this project was to create a protocol based on PPSP WG effort so that one single application could serve for different company “streaming channels”.

Standardization will be applied at different points of a Peer-2-Peer streaming transmission and on different nodes, in this project we focused on the Peer’s and tracker’s signalizing functionalities mostly.

PPSP can be applied at the client’s level. All p2p streaming applications no matter which, will work with the same protocol, a single application will suit to watch different company channels, which no longer will be making use of private protocols. At the present time each company/channel is working with its own protocols, which implies a concrete player for each kind of protocol.

Another interesting application is the possibility of using the same equipment for tracking, working as Bootstrap servers and helping on the distribution of a channel through long distances. Now a day’s companies have property applications for all those functionalities. With standardization, all companies will be able to share these applications making it easy for new channels to appear.

Source coding techniques will be applied too. This is done in order to optimize the streams quality to viewer’s capacities. ALTO protocol we will be used to select the partners each peer has, making our protocol robust and scalable.

Coolruc, developed by I2CAT, has been used as a reference of a P2P streaming application.
| Figure 1: Tracker interaction with peers. | 8 |
| Figure 2: Multiplexing video | 11 |
| Figure 3: Layered Encoding audio apart | 11 |
| Figure 4: Layered Encoding reserved channel for audio | 11 |
| Figure 5: Alto service & protocol scope. | 12 |
| Figure 6: ALTO network map | 13 |
| Figure 7: ALTO client embedded in P2P Tracker | 16 |
| Figure 8: ALTO Client embedded in P2P Client | 17 |
| Figure 9: ALTO Client embedded in P2P Client: Ranking | 17 |
| Figure 10: general Message | 19 |
| Figure 11: SubscriptionRequest | 21 |
| Figure 12: SubscriptionResponse | 21 |
| Figure 13 PublishChannelRequest | 22 |
| Figure 14: PublishChannelResponse | 22 |
| Figure 15: ListChannelsRequest | 23 |
| Figure 16: ListChannelsResponse | 23 |
| Figure 17: Join message | 24 |
| Figure 18: JoinResponse message | 25 |
| Figure 19: Leave message | 25 |
| Figure 20: LeaveResponse message | 25 |
| Figure 21: KeepAlive message | 26 |
| Figure 22: PartnershipMessage | 26 |
| Figure 23: PartnershipResponse | 27 |
| Figure 24: BufferMap Message | 27 |
| Figure 25: ChunkRequest | 28 |
| Figure 26: ChunkResponse | 28 |
| Figure 27: PeerDiscoveryMessage | 29 |
| Figure 28: Leave message | 30 |
| Figure 29: Peer Bootstrap Server Flow | 30 |
| Figure 30: Peer-Tracker flow | 32 |
| Figure 31: Peer-Peer flow | 32 |
| Figure 32: BufferMap | 33 |
| Figure 33: Layer transport video and audio | 34 |
| Figure 34: Hybrid MDC | 35 |
| Figure 35: Quality flows | 35 |
| Figure 36: bufferMap Chunk selection | 36 |
| Figure 37: P2P Streaming Architecture | 37 |
| Figure 38: P2p Streaming interfaces | 39 |
| Figure 39: Nested packages | 40 |
| Figure 40: Main UML Design | 41 |
| Figure 41: Bootstrap Server UML | 42 |
| Figure 42: PeerUML | 42 |
| Figure 43 Tracker UML | 42 |
| Figure 44: UML Operations | 43 |
| Figure 45: Application initial UML | 43 |
| Figure 46: Laptops | 44 |
Figure 48: Datacenter power conversion & distribution 45
Figure 49: Setting Bootstrap Server 54
Figure 50: Peer configuration 55
Figure 51: Bootstrap accepts a new peer 55
Figure 52: Setting peer as Tracker 56
Figure 53: Bootstrap accepts to publish the trackers channel 56
Figure 54: Channel published 56
Figure 55: Setting peer as client 57
Figure 56: Peer accepted by tracker 57
Figure 57: Partner selection 58
Figure 58: Accepting new partner 58
Figure 59: New partner 59
Figure 60: Chunks exchange 59
Figure 61: Peer leave channel 59
Figure 62: Message received by tracker when peer leaves channel 60

TABLES

Table 1 47
Table 2 48
CHAPTER 1. INTRODUCTION

Nowadays, there are quite an important number of companies which offer peer 2 peer streaming services, but each one uses property protocols which makes it impossible for costumers, applications and intermediary nodes to be shared between the companies who offer the media flows.

We have worked on a protocol to standardize p2p streaming, a protocol which will follow the indications given by the Internet Engineering Task Force who are currently working on a draft (PPSP WG).

In our proposal of protocol, we have designed a protocol which supports other aspects such as multiple description coding (layered coding e.g MDC and SVC) and uses ALTO protocol which are not included in the base protocol, but that will improve the flow between peers.

We have studied the indications given by the Internet Engineering Task Force referring to peer-to-peer streaming protocols. Research on how ALTO protocol and source coding techniques work and contribute to make PPSP protocol more optimal, robust and scalable.

This investigations have given us the knowledge to design our own protocol and to apply the improvements explained above.

We believe that implementing our protocol on a simple application will be helpful to represent how messages are exchanged between peers and truckers, and between peers and peers.
CHAPTER 2. PROBLEM STATEMENT

In this chapter we will tried to expose the actual context, limitations and problems related to video streaming, and how we have tried to solve them by making use of Peer-2-Peer streaming Protocol.

There are two basic methods to deliver streaming traffic through the Internet: Client-Server and the peer-2-peer (P2P) [P2P streaming Survey]. This project is based on the P2P streaming one, as we believe the server-client model presents a bottle neck in scalability due to server capacity limitations and Bandwidth consumption.

Video streaming is the main contributor for global IP traffic growth in the following years, accounting for 50% of the total traffic.

Peer 2 Peer streaming scalability is not limited by the server capacity because each consumer will become consumer and supplier at the same time, increasing the number of servers from where a future client will be able to stream content form.

Some consolidated examples of P2P applications are: PPLive, PPstream, UUSee, CNN (OctoShape).

Creating a Standard Peer-2-Peer streaming protocol: Proprietary P2P streaming protocols do not integrate well with infrastructure devices such as caches and other edge devices.

PPSP will serve as an enabling technology, building on the development experiences of existing P2P streaming systems.

Most Proprietary P2P streaming protocols differ in signaling transactions. Consequently, in order to support P2P streaming, the infrastructure devices need to understand and keep updated with various proprietary P2P streaming protocols. This introduces complexity and deployment cost of infrastructure devices.

Standardization allows effective integration with edge infrastructures such as cache and mobile edge equipment.

With standardization technical feasibility will be given to the peer-2-peer protocol and programs who work with it. Strong similarity with major systems which already use Tracker-based architecture and similar tracker and peer communication process, will help upcoming program designers to adapt easily to the new standard.

By creating a standard protocol and thus, more network operators will start deploying open and competitive Internet streaming service. Operators will be able to run their own streaming service or cooperate with P2P streaming vendors.

For user’s convenience, unified peer/client software may be developed, specially orientated for mobile devices. Storage and memory is constrained in
this kind of devices and avoiding the installation of several softwares carrying out the same function is a very interesting matter to consider.

**Protocols to be standardized:** We aim to standardize the information exchanged in the signaling protocols among different devices in P2P streaming systems. In this project we propose to standardize protocols in PPSP which enable the tracker communication and the peer communication components in the communication layer.

**PPSP protocol includes:**

- PPSP tracker protocol, a signaling protocol between PPSP trackers and PPSP peers.
- PPSP peer protocol, a signaling protocol among PPSP peers.

### 2.1 The PPSP tracker protocol

This protocol defines:

1) Standard format/encoding of information between PPSP peers and PPSP trackers, such as peer list, content availability, streaming status including online time, link status, node capability and other streaming parameters.

2) Standard messages between PPSP peers and PPSP trackers defining how PPSP peers report streaming status and request to PPSP trackers, as well as how PPSP trackers reply to the requests.

### 3.2.2 PPSP peer protocol

This protocol defines:

1) Standard format/encoding of information among PPSP peers, such as chunk description.

2) Standard messages among PPSP peers defining how PPSP peers advertise chunk availability to each other, as well as the signaling for requesting the chunks among PPSP peers.

**Common service types supported by P2P streaming systems are:**

- live streaming, all PPSP peers are interested in the media coming from an ongoing event, which means that all PPSP peers share nearly the same streaming content at a given point of time. In live streaming, some PPSP peers may store the live media for further distribution, which is known as TSTV (time-shift TV) where the stored media are separated into chunks and distributed in a VoD-like manner.
- **VoD**, different PPSP peers watch different parts of the media recorded and stored during a past event. In this case, each PPSP peer keeps asking other PPSP peers which media chunks are stored in which PPSP peers, and then pulls the required media from some selected PPSP peers.

### 3.2.3 China as a reference

Consider a popular program, for example the Chinese Spring Festival, where a P2P streaming provider is providing a live media broadcast from China. With existing deployments today, there is very little difficulty in watching the broadcast on the Internet from within China - this is already widespread practice. However if a viewer outside of China wants to watch the gala from outside China, they may have difficulties with smooth playback because of the insufficient number of peers outside of China, instability of dynamic peers, which makes meeting condition more difficult and perhaps a non stable end-to-end bandwidth assurance from the source to peers from outside China.

PPSP helps by ensuring that the providers have a common protocol to communicate. This reduces the case by case negotiation between the original provider and multiple CDN providers (from outside China, referring to the example we have above) if otherwise proprietary protocols are used makes it easier for both sides to interoperate.
CHAPTER 3. GENERAL REQUIREMENTS

As it is stated on P2P Streaming Protocol Draft, *P2P Streaming Protocol (PPSP) Requirement draft-zong-ppsp-reqs-04* (see web page reference [1]), PPSP defines the following requirements:

### 3.1 Locating and Connection

A peer will obtain a peerlist from the Tracker, including Peer IDs, Peer IP Addresses or other peer identities. The requirements stated by ppsp at this point are that the client must be able to locate and contact the peers given in the peerlist with minimal assistance, or none at all, from the Tracker.

By locating we mean that a peer can find a path between itself and the remote peer, on which messages can pass through. Trackers having public addresses may seem a seductive method to link different peers, but we must consider that by doing this, the trucker will probably become the bottleneck of our system. This is the reason why peer to peer communication through tracker is discouraged.

PPSP strongly recommends to implement a method where peers may locate and contact other peers without the trackers assistance, in our proposal we are using the peers IP addresses (see the implementation developed).

This point also includes the fact that Peer Protocol must provide a mechanism for peers behind different NAT/Firewall to connect with each other.

### 3.2 Information Exchange

This requirement is used to define what information peers will exchange to know where to download content from.

In many peer to peer streaming protocols such as PPLive and PPStream, a Client can obtain peerlist from both Tracker and remote peers. The trackers main function is to record the peers in a same swarm, but it may not be practical to include all of them (especially when we have a very high number of peers in a swarm) in the peerlist send to the client. In this case just some of them will be included. In our proposal, the selection of the peers is determined by ALTO parameters.

Once the client has received a first peerlist from the Tracker, and a partnership relation has been established with some of them, the ppsp protocol must provide a method from which a peer can get more peerlist, if he wants to. So, a peerlist request and a peerlist response carrying the peer list must be included in ppsp protocol, but its utilization may be optional.

Peer Protocol should enable peers to indicate the number of peers to be included in the peerlist when requesting peerlist from remote peers.
3.3 Property Exchange

Property parameters describe the peer’s connection properties, such as adsl connection for example, is the peer a computer or a mobile phone, physical location of the peer among others, etc.

Each peer must indicate its property parameters when requesting for a peerlist. With this functionality, a peer can select peer candidates depending on the property parameters obtained from the Tracker or other remote peers. In our protocol design this Property Exchange will be included with the ALTO parameters of each peer.

To make the whole system more effective and reach a higher performance, we should try to take full advantage of the low performance peers, as well as the high performance ones. Therefore a peer can indicate its own property expressing context features or node capacities, remote peers will then choose peers with similar property to be added to their peer lists.

3.4 Data Availability

PPSP must enable peers to request for Data Availability from remote peers. In our protocol this is achieved with the BufferMap Exchange Message, the BufferMap is a matrix send by a peer to all his established partners in which the available chunks are represented.

Data Availability is presented with a bitmap in some applications to exchange chunks availability, but it’s not the only way, in our design a BufferMap is used instead.

Peerlist exchanged among peers may also be different formats, PPSP Protocol must be agnostic of the specification of different applications.

3.5 Connection Status

Requirement influenced by Bittorrent Specification.

3.5.1 Application-level Congestion Control should enable a peer to whether it would like to upload content to the remote peers or not. This is especially useful when a peer is overloaded, when a new peer connects to it and request for further action the overloaded peer may make use of this feature and advertise to the demander that he is overloaded and he will not accept more requests.

3.5.2 Real-Time Features. Peer Protocol must support synchronous downloading/uploading. This requirement means a peer can download from multiple peers, or upload to multiple peers at the same time. Peer protocol must
provide corresponding mechanism to eliminate or decrease the influence of Jitter and packet loss.

3.6 Transportation Negotiation

Peer protocol must be able to negotiate a transportation protocol that both peers can support, UDP and TCP for example.

Peers must try to negotiate a transportation protocol and then exchange signaling messages to set up a transportation connection between them.

3.7 Security

Peer Protocol must guarantee peers' privacy. Peer Requests must only be received by the targeted remote peers. PPSP protocol must prevent attacks such as man in the middle or messages from being wire tapped.

A Peer should be able to validate the authentication of remote peers.

These are the Peer to Peer Protocol requirements, our objective is to extend these to a layered media or a multiple description coding as it is described in the layered media chapter.

The requirements described above have been obtained from the P2P Streaming Protocol (PPSP) Requirements draft-zong-ppsp-reqs-04 (see web references [1]).
CHAPTER 4. STATE OF ART

4.1 Protocol functionalities

In this chapter we are going to expose P2P streaming basic functionalities.

P2P streaming functionalities include:

- Maintaining information about which peers are connected to which swarm in some directory service.

- In each swarm, exchange of information about content availability will take place among peers, or between tracker and peers. The exchange of information will include for example which chunks are stored by a peer.

- In each swarm, exchange of the actual content among peers takes place. As shown in Figure 1, the common information flows in a P2P streaming system in each context include:

Context:

**When a peer wants to receive streaming content:**

1) Peer acquires a list of peers in the swarm from the tracker. A swarm can be indexed by a channel ID, streaming file ID, etc.
2) The peer exchanges its content availability with the peers obtained from the peer list.
3) Client identifies the peers with desired content and requests it from them.

**When a peer wants to share streaming content with others:**

The peer sends information to the tracker about the swarms he belongs to, plus streaming status and/or content availability.

Diagram:

![Tracker interaction with peers](image)

Figure 1: Tracker interaction with peers.

See [3] for more references to ppss protocol functionalities.
4.1.1 Peer-2-Peer Streaming feature

Peer-2-Peer streaming will ensure in-time and continuous streaming delivery and a limited start-up and transmission delay. 1.5Mbps rates giving TV quality images and 400kbps through internet, are rates easily obtained using PPSP.

Other functionalities that we have considered when designing the protocol are that the protocol must be scalable and robust.

Scalable to a high number of viewers, the architecture of our protocol has been designed to support unexpected high numbered viewer incorporations. For example the number of viewers a sports channel will have to support on a final will be much higher than on a standard match.

ALTO protocol plays a relevant role here too, as it will be constantly called to check if new incorporations have better characteristics than the ones we already are downloading streams from.

PPSP protocol is robust to sudden connection falls or to nodes disconnection. If a path between nodes falls and some chunks are lost or reach us latter than expected as a consequence of the new route, PPSP signalization allows to react fast enough so that the out image flow is merely affected. Same will happen if a node from who we are receiving chunks suddenly loses its connection or simply disconnects without “warning”. Thanks to signalization, it will be possible to react very rapidly; the protocol will choose automatically a new active partner from whom to download the chunks within the time the buffer has not emptied yet.

4.2 Source coding for layered streaming: MDC and SVC

Multiple Description coding and layered streaming are used to solve the problem of having to transmit a different multimedia flow depending on the client’s connection quality. Let’s suppose we have 2 clients who want to watch the same channel, one on a PDA and the other on a personal computer. The PDA client has a much slower connection than the personal computer one, so he could not process correctly the same flow that is send to the computer. By making use of a layered streaming it is not necessary to send a lower bit rate (with less quality) to the PDA and a faster one to the computer.

Layered streaming permits to send a basic layer containing the most significant data and which would suite the slowest connections. Enhancement layers will be added to improve quality from the base video. The enhanced layers would be processed only by those hosts with a speed connection that would permit to do so, the computer in our example.

By using Layers not only a single flow is transmitted to all users, but peers working on different versions will be able to help each others. The received flows may be redistributed to other peers with out mattering the connection and
flow version each one has. This will help to reduce significantly the overall transmission cost.

4.4.1 Multiple Description Coding (MDC)

MDC is a source coding technique which encodes a signal (this case a media resource: audio/video) into a number of N different sub-bitstreams (where N≥2).

The source information will be encoded into multiple descriptions which will be transmitted through different flows. When any of the descriptors, which are all independently decidable, are available at the receiver, a reconstruction of the source can be obtained.

The quality of the reproduced media is proportional to the number of descriptors received, that is; the more descriptors are received, better the reconstruction quality is. The idea of MDC is to provide error resilience to media streams. Since an arbitrary subset of descriptors can be used to decode the original stream, network congestion or packet losses, which are common in best-effort networks such as the Internet, will not interrupt the reproduction of the stream (continuity) but will only cause a (temporary) loss of quality. (see Books, memories and drafts reference [1])

Reconstructing a source with acceptable quality in the absence of one description at the receiver requires the two descriptions to be adequately correlated.

The higher the correlation, the lower the difference between central and side distortion. Therefore, in MDC, source coding efficiency will be sacrificed and redundancy is increased.

4.2.2 Scalable Video Coding (SVC)

SVC, also called Layered Coding adapts the video information to the network constrains splitting the images into different layers (similar to MDC). The quality of the image will be determined by the number of layers. From the base layer, each successive layer improves the image quality, getting the full picture quality with the total amount of layers used.

SVC descriptors are dependent, which means that no descriptor may be lost in order to reconstruct the image.

4.2.3 Implementing Multiple Description Coding

The biggest drawback of implementing Multiple Description Coding it’s the extra over head that will be added to each layer when multiplexing.

The overhead must be added to correctly synchronize the chunks obtained from the different flows back to a high definition image single flow.
Each additional line improves the video quality, the number of lines the receptor will process will vary depending on his connection properties. Columns will increase proportionally to the reproduction time.

The audio chunks will travel on a different flow and will be treated on a separated buffer from the video matrix.

We have chosen this method because we believe it is the option with best scalable video qualities. One single original video flow will suit different connection speeds presented by destination nodes.

By making use of a single flow and not one for video and another for audio, we simplify the handling of the flows and the synchronization, which finally leads to a smaller overhead.
See [1, 2] books, memories and drafts to read the reference to layered streaming.

### 4.3 ALTO protocol

The ALTO Service intends to provide a simple way to convey network information to applications.

ALTO Service provide information such as preferences of network resources with the goal of modifying network resource consumption patterns while maintaining or improving application performance.

P2P applications can take advantage of the ISP’s knowledge to avoid bottlenecks and boost performance.

#### 4.3.1 ALTO Service and Protocol Scope

A network region in this context can be an Autonomous System, an ISP, perhaps a smaller region, or perhaps a set of ISPs; the details depend on the deployment scenario and discovery mechanism. Alto will present its “internet view” of this region.

![Figure 1: Basic ALTO Architecture.](image)

![Figure 5: Alto service & protocol scope.](image)
ALTO Information is provided via the ALTO Protocol by answering the following four types of queries:

**Server Capability:** It lists the details on the information that can be provided by an ALTO Server.

**Endpoint Property:** Given an endpoint, it gives the set of network-aware properties associated with the endpoint. An example endpoint property is its Network Location property or connectivity type (e.g., ADSL, Cable, or FioS).

**Reverse Property Map:** It is the reverse of the preceding. In particular, given a property, it lists the endpoints with the given property.

**Path Property Lookup:** It gives information on the properties of paths among Network Locations.

### 4.3.2 ALTO main characteristics

**Network MAP:** The Network Location endpoint property allows an ALTO Server to group endpoints together to indicate their proximity. The resulting set of groupings is called the ALTO Network Map.

The network map may be used to communicate simple preferences. For example, an ISP may prefer that endpoints associated with the same Point-of-Presence in a P2P application communicate locally instead of communicating with endpoints in other locations.

**Network Map parties:** PID: PID is an identifier providing an indirect and network agnostic way to specify a network aggregation. For example, a PID may be defined (by the ALTO service provider) to denote a subnet, a set of subnets, a metropolitan area, a PoP (point-of-presence), an autonomous system, or a set of autonomous systems.

The following figure summarizes these concepts.

![ALTO Network Map](image)

Figure 6: ALTO network map.
Endpoint PID Query: The Endpoint Property query against the Network Map allows an ALTO Client to retrieve the PID of a given endpoint.

Reverse Network Map Query: The Reverse Property Map query for Network Map allows an ALTO Client to obtain a map from PIDs to lists of endpoints: for each PID, the map includes its list of endpoints.

Path rating: Path Rating is based on Path Cost, which conveys the preference of an ALTO Server on communication among Network Locations. Path Costs have attributes too.

Type: identifies what the costs represent.
Mode: identifies how the costs should be interpreted (numerical or ordinal interpretation).

For example, it is possible for an ALTO Server to return a set of IP addresses with costs indicating a ranking of the IP addresses.

Design Approach: Alto uses a lightweight XML encoding, with the goal of leveraging current HTTP implementations and infrastructure. ALTO messages are denoted with HTTP Content-Type "application/aldo".

4.3.3 Use of Existing Infrastructure

ALTO may take advantage of the huge installed base of HTTP infrastructure, including HTTP caches, mature software implementations for both HTTP and XML. Many P2P clients already have an embedded HTTP client, and authentication and encryption mechanisms in HTTP and SSL which could be reused.

4.3.4 Message Format

ALTO Protocol uses a restful design operating over HTTP. Both Query and Response follow the standard format for HTTP Request and Response messages.

- **Query Message**:
  Method: Indicates operation requested by the ALTO Client (along with URI Path).
  URI Path: Indicates the operation requested by the ALTO Client.
  URI Query String Parameters: Indicates parameters for the requested operation.
  Headers: Indicates encoding of the Body.
  Body: Indicates additional request parameters that are not concisely representable as Query String parameters.

- **Response Message**:
  Status Code: Indicates either success or an error condition.
Headers: Indicates encoding of the Body and caching directives. Body: Contains data requested by the ALTO Client.

Example of XML Document Carried by ALTO Protocol Messages

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?oxygen RNGSchema="config-schema.rnc" type="compact"?>
<alto xmlns="urn:ietf:params:xml:ns:p2p:alto">
...
</alto>
```

4.3.5 Processing

Client Processing: The ALTO Client selects a specific ALTO Server to communicate with and establishes a TCP connection. The ALTO protocol on top of this TCP connection can be secured through SSL/TLS. Finally the client sends it down the TCP/IP stack.

Server Processing:

Errors:
- Invalid Query Format: returns 400
- Unsupported Query: returns 501

The supported queries are:

- Server Capability
- Endpoint Property Lookup: allows an ALTO Client to query for properties of Endpoints known to the ALTO Server.
- Reverse Property Lookup: allows an ALTO Client to query for Endpoints with common properties.
- Path Rating Lookup: allows ALTO Clients to query for Costs amongst Network Locations.

4.3.6 Useful Cases

ALTO Client Embedded in P2P Tracker: (OUR CASE)

P2P applications use Trackers to manage swarms and perform peer selection. Acting as an ALTO Client, a P2P tracker can use ALTO information as an additional information source to enable more network-efficient traffic patterns and improve application performance.

A P2P tracker can obtain and locally store ALTO information (the Network Map and Cost Map) from the ISPs containing the P2P clients, and benefit from the same aggregation of network locations done by ALTO Servers.
Work flow:

1. The P2P Tracker requests the Network Map covering all PIDs from the ALTO Server using the Reverse Property Lookup query. The Network Map includes the IP prefixes contained in each PID, allowing the P2P tracker to locally map P2P clients.

2. The P2P Tracker requests the Cost Map amongst all PIDs from the ALTO Server.

3. P2P Client joins the swarm, and requests a peer list from the P2P Tracker.

4. The P2P Tracker returns a peer list to the P2P client. The returned peer list is computed based on the Network Map and Cost Map returned by the ALTO Server, and possibly other information sources.

   * is possible that a tracker may use only the Network Map to implement hierarchical peer selection by preferring peers within the same PID and ISP.

5. The P2P Client connects to the selected peers.

Other options:

ALTO Client Embedded in P2P Client

P2P clients may also utilize ALTO information themselves when selecting from available peers.

When a P2P Client uses ALTO information, it typically queries only the ALTO Server servicing its own ISP. A my-Internet view provided by its ISP's ALTO Server can include preferences to all potential peers.

The following example is a use case where a P2P Client locally applies ALTO information to select peers.
Figure 8: ALTO Client embedded in P2P Client

1. The P2P Client requests the Network Map covering all PIDs from the ALTO Server servicing its own ISP.

2. The P2P Client requests the Cost Map amongst all PIDs from the ALTO Server. The Cost Map by default specifies numerical costs.

3. The P2P Client discovers peers from sources such as Peer Exchange (PEX) from other P2P Clients, Distributed Hash Tables (DHT), and P2P Trackers.

4. The P2P Client uses ALTO information as part of the algorithm for selecting new peers, and connects to the selected peers.

ALTO Client Embedded in P2P Client: Ranking

The ALTO Client gathers a list of known peers in the swarm, and asks the ALTO Server to rank them.

Figure 9: ALTO Client embedded in P2P Client: Ranking

1. The P2P Client discovers peers from sources such as Peer Exchange (PEX) from other P2P Clients, Distributed Hash Tables (DHT), and P2P Trackers.
2. The P2P Client queries its ALTO Server, including discovered peers as the set of Destination Network Locations, and indicates the ‘ordinal’ Cost Mode. The returned Cost Map indicates the ranking of the candidate peers.

3. The P2P Client connects to the peers in the order specified in the ranking.

4.3.7 Reflection of ALTO usage

The application must select which peers to use based on the ALTO parameters and other sources of information. With this in mind, the usages of ALTO costs are intentionally flexible, because different applications may use the information differently. For example, an application that connects to just one address may have a different algorithm for selecting it than an application that connects to many.

In addition, the application might account for robustness, perhaps using randomized exploration to determine if it performs better without ALTO information.

Factors that may determine the peers selection:

- Prefer network locations with lower ordinal rankings (i.e., higher priority)
- Use bandwidth matching (e.g., at an Application Tracker) and choosing solution (within bound of optimal) with minimal network cost.
CHAPTER 5. PROTOCOL SPECIFICATION AND FUNCTIONALITIES

5.1 Message specification

Messages used

PPSP messages are exchanged among:

- Bootstrap Server and Peers
- Tracker and Peers
- Peers

In all cases the messages exchanged use the basic format of RFC 2822. PPSP messages will consist of a start-line, one or more header fields, an empty line indicating the end of the header fields, and an optional message-body. The idea here is that all the messages inherit from a Message class in order to abstract the lower layers from the message types.

All the messages inherit from a Message class in order to abstract the lower layers from the message types. Message will be composed as follows and may be used in the following contexts:

Message: This is the standard message from where all other messages will inherit. The massage will be included in the body of a standard IP which will includes source address, destination address and port of the nodes involved, these messages will be transported over UDP or TCP.

+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+
<table>
<thead>
<tr>
<th>Version</th>
<th>Method</th>
<th>Length</th>
<th>Status code</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>--------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>Body</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+

Figure 10: general Message.

Version: The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capability for understanding further communication, rather than the features obtained via that communication. No change is made to the version number for the addition of message components which do not affect communication behavior or which only add to extensible field values.

Version field is 8 bits. The first 4 bits represent <major> number and the rest 4 bits represent <minor> number.
Method: (it can be splitted in two fields: X and type) the method parameter will determine two important factors, the type of message and the method related to it.

Three types of possible messages exist, a Client-Bootstrap-Server message, a Client-Tracker message and a message between peers (clients).

The method parameter will be represented by 8 bits. The first 2 bits will stand for the type of message and the rest (6bits) will determine the action to be carried out by the node.

00 -> Client-Bootstrap-Server message.
01 -> Client-Tracker message.
10 -> Peer-Peer message.
11 -> reserved.

*On the following sections we will specify the values taken by the resting 8 bits under each type of message.*

**Length:** Message Length (16 bit) indicates the length in Bytes of the message, including the message header and the following variable message body.

**Status code:** The status code will reflect the general status of a request or response message. The status code may take the following values:

- Successful (OK): 0x1
- Invalid syntax: 0x2
- Payment required: 0x3
- Request forbidden: 0x4
- Object not found: 0x5
- Internal server error: 0x6
- Does not support: 0x7
- Temporarily overloaded: 0x8
- Version not support: 0x9

*Reserved:* The reserved camp is a free space which may be usable in further versions of the protocol.

**Body:** The message body will carry the variable message specific to each type and to each action to be carried out.

**5.1.1 Client – Bootstrap Messages**

Each message will have its representation in the last 8 bits included on the method parameter found in the general message. These will always be preceded by the 00x code.

The following methods may be used between a client-bootstrap server and a peer communication:
SubscriptionRequest. The subscriptionRequest will be represented by the 000001 code. This message is used by the client to login to the bootstrap Server and grant access to its database, where channel and tracker information will be kept. When logging to the bootstrap server, this will update its connected peers list.

Message format:

```
+-----------------+-------------------+-------------------+-------------------+-------------------+
| Version          | 00000001          | Length            | Status code       | Reserved          |
+-----------------+-------------------+-------------------+-------------------+-------------------+
| Peer ID          |                   |                   |                   |                   |
```

![Figure 11: SubscriptionRequest](image)

Peer ID will be the parameter send by the client to indentify itself against the server.

Peer ID will be defined in a maximum space of 20 Bytes.

SubscriptionResponse: It is the response to SusbscriptionResquest and it will be represented by the following code in the method field 00000010. The response represented by Response ID, will change depending on the situation:

Response codes:

0x0 = OK
0x1 = User found, wrong password
0x2 = User not found
0x3 = Server is full

Message format:

```
+-----------------+-------------------+-------------------+-------------------+-------------------+
| Version          | 00000010          | Length            | Status code       | Reserved          |
+-----------------+-------------------+-------------------+-------------------+-------------------+
| Response ID      |                   |                   |                   |                   |
```

![Figure 12: Subscription Response](image)

PublishChannelRequest: The PublishCannelRequest will be represented by the 00000011 code. This message is used by a peer which wants to register a channel into the bootstrap server. For registering a channel, it is needed the following data:
- Channel name
- Channel description
- Channel’s tracker information

Message format:

```
+-----------------------------------------------+-----------------------------------------------+
|                               |                                               |
| Version: 00000011               | Length: Status code: Reserved                |
+-----------------------------------------------+-----------------------------------------------+
| channel name, channel description, channel tracker info |
+-----------------------------------------------+-----------------------------------------------+
```

Figure 13 PublishChannelRequest

**PublishChannelResponse**: It is the response to publishChannelRequest and it will be represented by the 00000010 code in the method field. The response represented by Response ID, will change depending on the situation:

Response codes:

- 0x0 = OK
- 0x1 = No permission
- 0x2 = User not found
- 0x3 = Server is full
- 0x4 = Channel name already used

Message format:

```
+-----------------------------------------------+-----------------------------------------------+
|                               |                                               |
| Version: 00000100               | Length: Status code: Reserved                |
+-----------------------------------------------+-----------------------------------------------+
| Response ID                        |
+-----------------------------------------------+-----------------------------------------------+
```

Figure 14: PublishChannelResponse

**ListChannelsRequest**: This message is sent by a Peer in order to retrieve a channel list from the bootstrap server. The representation for this method will be 00000101.

By making use of the searchRequest parameter, the client will send a XML description of the channels name or of the tags the channels included on the response list must include.
Message format:

```
+---------------------+-----------------+--------------+---------------+---------+
|          Version     |      00000101   |     Length   |    Status code|  Reserved|
+---------------------+-----------------+--------------+---------------+---------+
| Peer ID             |                 |              |               |         |
+---------------------+-----------------+--------------+---------------+---------+
| SearchRequest       |                 |              |               |         |
+---------------------+-----------------+--------------+---------------+---------+
```

Figure 15: ListChannelsRequest

ListChannelsResponse: this is the answer to the SearchRequest message. ListChannelsResponse contains a list of channels registered at the bootstrap server. The method code for this massage is 00000110.

Once the bootstrap server has compared the information contained in the SearchRequest parameter with its channel list database, he will answer with a ChannelList parameter.

ChannelList parameter contains a list of channels that have been found to share the tags given by the client. The method code for this massage is 00000110.

Message format:

```
+---------------------+-----------------+--------------+---------------+---------+
|          Version     |      00000110   |     Length   |    Status code|  Reserved|
+---------------------+-----------------+--------------+---------------+---------+
| ChannelList          |                 |              |               |         |
+---------------------+-----------------+--------------+---------------+---------+
```

Figure 26: ListChannelsResponse

5.1.2 Client – Tracker Messages

The client-Tracker communication will be used to communicate the candidate’s status and to request partner lists form the tracker.

ALTO protocol may be used to optimize the partner list given to the client.

The camp referred as ALTO Parameters will allow the tracker to make use of the Alto protocol* optimizing the partnership process. The first two bits from the method code will always be set to 01 to identify a client-tracker communication.
Join message: Message send by the client to the tracker to express his desire to join a channel. This message is send by the peer to the tracker in order to obtain a list of peers who are watching the same channel.

The last 6 bits from the method code to identify this message will be set to 000001.

Message format and values:

<table>
<thead>
<tr>
<th>Version</th>
<th>01000001</th>
<th>Length</th>
<th>Status code</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 17: Join message:

Peer ID, it’s the client’s identification, by using the clients ID the tracker will determine weather the client may or may not join its channel.

If a client joins a channel the client ID will be added to the trackers database.

ALTO parameters, this will be send so that the tracker may choose the best partner candidates from its database for a concrete client. ALTO parameters may be determined by BW, geographical area, etc.

*If the trackers database has not enough entry’s to choose from, the ALTO parameters will be ignored.

Join Response message: Response massage to Join message, where a selected list of nodes who are watching the same “channel” the peer wants view will be send.

The method code for this message is 01000010.

A response ID code will be send as well, where the values taken will be:

0x0 = OK
0x1 = No permission
0x2 = Channel is full

Message format:
Partner List: list of peers who are watching the same program.

**Leave message:** Leave message is sent from peer to Tracker to announce it is going to leave the system.

The method code for this message is 01000011

When the tracker receives this message, he deletes the peer’s ID from its database and responses with a LeaveResponse.

**Message format:**

```
+-----------------+-----------------+---------------+-----------------+----------+
| Version         | 01000011        | Length        | Status code    | Reserved |
+-----------------+-----------------+---------------+-----------------+----------+
| Peer ID         |                 |               |                 |          |
+-----------------+-----------------+---------------+-----------------+----------+
```

**Figure 19: Leave message**

**LeaveResponse message:** This is the response to the Leave message, it will be represented by the 01000100 method code.

**Message format:**

```
+-----------------+-----------------+---------------+-----------------+----------+
| Version         | 01000100        | Length        | Status code    | Reserved |
+-----------------+-----------------+---------------+-----------------+----------+
|                 |                 |               |                 |          |
+-----------------+-----------------+---------------+-----------------+----------+
```

**Figure 20: LeaveResponse message**

**KeepAlive message:** Messages send by peers to the tracker every established period to advertise they are still connected to the system.

The method code to identify a KeepAlive message is: 01000101
If a tracker does not receive a keepAlive message within the established time period from a peer, its counter expires and his peer ID entry is deleted from the trackers database. This situation will provoke a Peer to send a new JoinRequest message to the tracker.

Message format:

<table>
<thead>
<tr>
<th>Version</th>
<th>Length</th>
<th>Status code</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Peer ID

Figure 21: KeepAlive message

5.1.3 Peer – Peer Messages (Peer Protocol)

The messages used between the different peers can be classified as request messages or response messages.

The two first bits from the method code that will always precede the Peer-Peer messages will be 10.

PartnershipMessage: This message is sent by a peer to other discovered peers in order to establish a peer relation.

The aim of this message is to establish a partnership relation between peers so they may exchange chunks between each other.

ALTO parameters are sent too in case the destination peer wants to determine if the source peer will be a good partner or not. The last six bits from the method code will be set to 000001.

Message format:

<table>
<thead>
<tr>
<th>Version</th>
<th>Length</th>
<th>Status code</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Peer ID

ALTO parameters

Figure 22: PartnershipMessage
PartnershipResponse: This message is the response to the PartnershipMessage and it will include a response code which stands for:

0x0 = OK
0x1 = No permission, no partnership relation will be established.
0x2 = Partner relations full.

PartnershipResponse will be represented by the 10000010 method code.

Message format:

```
+-------------------------------+-
|         |         |           |         |
| Version | 10000010 | Length    | Status code |
+-------------------------------+-
|         |         |           |         |
|         |         |           | Reserved |
+-------------------------------+-
|                                 |
|                                   |
|                             Response ID |
|                                   |
+-------------------------------+-
```

Figure 23: PartnershipResponse

When a partner is accepted both peers will update their established partners list.

BufferMap Message: This message includes a representation of the chunks that a Peer has in its Buffer in the form of a Buffer Map. This message will be sent to all the partners a node have.

The method code for this message is 10000011.

From the bufferMap Matrixes received, the peers may determine from which peer they may request the chunks themselves are missing.

From the received bufferMap Matrixes, the peers may determine from which peer they may request the chunks themselves are missing according to the scheduling algorithm that a peer application implements.

Message format:

```
+-------------------------------+-
|         |         |           |         |
| Version | 10000011 | Length    | Status code |
+-------------------------------+-
|         |         |           |         |
|         |         |           | Reserved |
+-------------------------------+-
|                                 |
|                                   |
|                             Peer ID |
|                                   |
+-------------------------------+-
```

```
+-------------------------------+-
|                                 |
|                                   |
|                             BufferMap |
|                                   |
+-------------------------------+-
```

Figure 24: BufferMap Message
Further information on the Buffer Map implementation may be found on the Buffer Map section.

**ChunkRequest:** This message will be send by the nodes to their partners to ask for missing chunks.

ChunkRequest messages last six bits will be set to 000100.

**Message format:**

```
+------------------+-+------------------+-+------------------+-+------------------+-+
|   Version        | 10000100  | Length           | Status code      | Reserved         |
+------------------+-+------------------+-+------------------+-+------------------+-+
  Peer ID
+------------------+-+------------------+-+------------------+-+------------------+-+
  ChunkList
```

Figure 25: ChunkRequest

ChunkList contains the chunkIDs a peer is requesting for.

**ChunkResponse:** This message is the response from one partner to a ChunkRequest, and it contains the chunks that have been requested and the sender's ID.

ChunkResponse messages have the following method code: 10000101.

*The maximum number of chunks contained in a ChunkResponse message will be limited by the system*

**Message format:**

```
+------------------+-+------------------+-+------------------+-+
|   Version        | 10000101  | Length           | Status code      |
+------------------+-+------------------+-+------------------+-+
  Source ID
+------------------+-+------------------+-+
  Chunk 1
+------------------+-+------------------+-+
  Chunk 2
```

Figure 26: ChunkResponse
PeerDiscoveryMessage (Membership exchange): This message contains a list of the known peers of the local peer.

Messages including the local partners list will periodically be sent between partners.

The aim of these messages is to dynamically update the partners list with new possible members.

ALTO parameters can also be included so that the nodes may consider establishing or not a new partnership relations with new peers included in the gossip messages.

The method code included in these messages will be 10000111.

The PeerDiscoveryMessage will be used as a “keepAlive” message too. If a node does not receive from an expected partner a PeerDiscoveryMessage within the established time period, its counter expires and his peer ID entry is deleted from the peers established partner list.

Message format:

```
<table>
<thead>
<tr>
<th>Version</th>
<th>10000111</th>
<th>Length</th>
<th>Status code</th>
<th>Reserved</th>
</tr>
</thead>
</table>
```

Source ID

GossipPeerList

ALTO parameters

Figure 27: PeerDiscoveryMessage

Leave message: Leave message is sent from a peer to all its established partners to announce it is going to leave the system.

The method code for this message is 10001000
When a partner receives this message, he deletes the peer’s ID from its established peerlist.
5.2 MESSAGE FLOW

5.2.1 Peer - Bootstrap Server

In the SearchResponse Message the client will obtain the ID of the Trackers who are in charge of the channels sharing the tags given by the client.

Once the client contacts the channel source (Tracker) this will address him to a peer group.
5.2.2 Peer – Tracker

1. The Peer Contacts the tracker in charge of the channel he is interested in. AltoParameters will describe the peers “situation and context”.

2. The tracker updates its database with the new Peers ID and responds accordingly with a list of nodes who are watching the same channel and who best suit the new peer according to its ALTO parameters.

3. KeepAlive messages are sent with a given period from the client to the tracker. If the tracker does not get a keepAlive message within a given time, a counter starts, and when it expires the peersID is deleted from the trackers database.

LeaveMessages and LeaveResponse Message are used when a peer wants to announce his desire of leaving the system. When a peer announces he is leaving the system, the tracker deletes the leaving peer’s ID from its database.
5.2.3 Peer – Peer

Figure 30: Peer-Peer flow.

1. Client sends a PartnershipMessage to another peer obtained from the ListMembers who is watching the same channel. This is done to establish a partnership relation between peers so the may exchange chunks between each others.

2. The peer responds with a positive partnershipResponse and a partnership relation is established. Once a partnership relation is established, the client will update its established partners list.

3. A bufferMap message will be send to all the peers included in the clients database (established partners list). This message will contain a buffer map matrix where all the chunks the client has and the ones he does not have will be represented (further information on the buffer map chapter).

4. The client will select those partners who have the chunks he is missing (he will know that from the buffer map matrixes he is receiving from his established partners). And will ask for each missing chunk to the partner how it has it.
5. The missing chunks will be sent between partners using the chunkResponse.

6. GossipMessages are sent between the established partners to dynamically update the partners list with new possible members. ALTO parameters will be included so that the nodes may consider establishing or not a new partnership relations with new peers included in the gossip messages.

7. KeepAlive messages are sent with a given period between peers. If the counter related to a peer expires because no keepAlive message has been received from him, the peer’s ID is deleted from the database.

LeaveMessages are sent when a peer wants to announce his desire of leaving the system.

The messages design has been developed from the Tracker protocol, (see [4] Books, memories and drafts).

5.3 Buffer map exchange between Partners

In this chapter the buffer map Exchange between partners will be explained.

Each node will send periodically to all the nodes included in its partner list a matrix representative of the chunks the node already has and the ones which he is missing.

The Matrix will be called the buffermap, and it will constantly be updated accordingly to the dynamic window which will be moving through time along with the media being watch.

The following scheme helps to explain this concept:

![BufferMap exchange](image)

Figure 32: BufferMap exchange

As the media is reproduced the window moves along as well, this is done so that the chunks that were missing during a certain time of reproduction won’t be considered once that sequence has been watched.
We are just interested in obtaining the chunks that have not been reproduced yet, if we receive a chunk out of time we will not re-reproduce the scene so that the missing chunk is filled.

There are different ways to implement the idea we have just described. The simplest implementation would consist of a single line, where the video would be transmitted on a single layer, and another single line for the audio. In this context a unique quality of video and audio could be transferred.

<table>
<thead>
<tr>
<th>Video</th>
<th>Audio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 1 1 1</td>
<td>1 0 1 1 1 0 1</td>
</tr>
</tbody>
</table>

Figure 33: Layer transport video and audio.

The following option would be to multiplex high definition flow into lower quality flows (as it is reflected on the first figure). Each line of the matrix will represent a video streaming definition; depending on the desired quality (which will mostly be defined by the clients connection speed) the node will consider 2, 3, 4 or 5 lines.

The following option would be to multiplex high definition flow into lower quality flows (as it is reflected on the first figure). Each line of the matrix will represent a video streaming definition; depending on the desired quality (which will mostly be defined by the clients connection speed) the node will try to obtain a bigger or smaller number of lines, then the receiver will apply an upsampling process to regenerate the image flow. With more flows being upsampled, the receiver will obtain a better image quality.

Three basic balanced MDC techniques are used in the splitting process, and they function as follows:

**Spatial MDC:** The spatial technique consists in splitting each individual frame into several descriptors, working in the pixel-domain. Resultant descriptors will have a portion of the original spatial information. Additionally, it must be noticed that they will have lower resolution than the original resource, but they maintain the same aspect ratio. As it is obvious, each description has a lower bitrate in comparison with the original video resource. This process can be seen as a spatial subsampling.

**Temporal MDC** This technique consists in allocating each frame of a video sequence in a different descriptor. The easiest manner to implement this technique is using a round robin strategy, weighted or non-weighted depending on the needs.

**Hybrid MDC** This technique is based on a combination of spatial and temporal techniques.
Concretely, the considered hybrid technique sends one full frame in each description while in the rest of the descriptors it is sent a part of the original frame which is different from the other descriptors.

This technique introduces more redundancy in data in comparison with the other two techniques; however, the resulting descriptors are generated with a lower bit rate in comparison with the original video resource.

The process is presented in Figure 34.

![Figure 34: Hybrid MDC](image)

The client will effectuate an operation with its own buffermap against the ones he receives from his partners to determine whether a partner has chunks he is missing and if he has, the client will ask for them. The client will effectuate this operation each time he receives a new buffermap and at the same time he will actualize its own buffermap.

The client will send his actualized buffermap to all his partners so these may do the same operation and ask for any chunk the client has and they are missing.

The following scheme represents the chunk selection operation and the following request:
Peer to Peer Streaming Protocol Standardization

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Clients Buffermap

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Partner’s received buffermap

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

XOR Operation

Now the client will ask to the partner for the five chunks he is missing and the partner has.

**Figure 36: bufferMap Chunk selection**
PPSP architectures are defined as follows:

**Complete P2P streaming system:**

![Figure 37: P2P Streaming Architecture](image)

1. **Transport Layer** is responsible for data transmission between peers. UDP, TCP or other protocols can be used.

2. **Communication layer** includes three components:
   2.1) **Tracker communication** is a component that enables each peer to get peer list from the tracker and/or provide content availability to the tracker.
   2.2) **Peer communication** is a component that enables each peer to exchange content availability and request other peers for content.
   2.3) **Bootstrap** is a component that enables newly joined nodes to obtain tracker information.

3. **Information layer** is responsible for peer and content information collection and management.
   3.1) **Registration** is a component that enables nodes to register to the system, and publish the content information. The information may include but is not limited to: content description, content type, creation time, node information such as physical location, IP address.
3.2) Report is a component that enables peers to report streaming status to the tracker. The information may include peer inbound/ outbound traffic, amount of neighbor peers, peer health degree and other streaming parameters.
3.3) Statistics is a component that enables trackers to manage the aggregated system information for global control in upload bandwidth consumption, overhead consumption and other regards.

4) Play-out layer is responsible for controlling the action of media play (e.g. start, pause, resume, stop, fast-forward, and rewind).

5) Application layer is the top layer for streaming applications.

6.1 Function Entities and Interfaces

Let’s recall on how ppsp communication flows, a peer may get Tracker’s address by some way (form a bootstrap server in our implementation), and then send JOIN message to the Tracker. Tracker receives the request, assigns a peer ID and response with the peer ID to the requesting peer. Meanwhile, Tracker updates the peer status records. The peer has to send KEEPALIVE message periodically to notify he is still connected, or else Tracker will assume that the peer is offline and may remove the peer from content status and peer status.

When the peer wants to watch a live channel or VoD, it sends a GET message to the Tracker to show which content it is going to watch. Tracker replies to provide candidate peers list, from which the peer can download content. The peer can also announce the content it owns to the Tracker and Tracker the Tracker upon receiving PUT request will update content status to include the PUT peer into peer list.

In order to obtain the latest condition of peers, Tracker may send STATISTICS messages to peers and peers report with the requested information (this aspect has not been taken into account in our implementation). LEAVE message is sent from peer to Tracker to announce it is going to leave the system, and so the Tracker deletes the peer from his database.

The entities and interfaces participating in the P2P streaming systems are in the figure bellow.
Figure 38: P2p Streaming interfaces

Function Entities of PPSP Tracker Communication between Tracker and Peer entities support the following functions:

- Transmission of JOIN/LEAVE messages.
- Transmission of KEEPALIVE messages.
- Transmission of PUT messages, by which peers tell trackers what they have.
- Transmission of GET messages, by which peers request what they want and get candidate peers list from trackers.
- Transmission of STATISTICS requests and responses, by which trackers can get peers status, network performance, etc.
CHAPTER 7. IMPLEMENTATION ON A SIMPLE PROGRAM

7.1 Application Design (UML)

We have developed a simple program using java-Eclipse which simulates a streaming application that would make use of our PPSP protocol design.

This implementation will help us to understand in a much easier way how ppsp protocol works, which message flow does it follows and which data does each message contain.

The following UML design generated with OMONDO represents our project:

Nested packages diagram:

![Nested packages diagram](image)

Figure 39: Nested packages

As you can see from the previous diagram, the program is divided into four main blocks, initial corresponding to the configuration panel, where we will be able to set our program as Bootstrap Server or as a Peer.
The bootstrap and peer blocks contain all the specific functions according to each working mode. The peer block also contains the peer.Client and the peer.Tracker configurations.

The last block, operations contains the basic and common operations such as reading from files, write into file, and convert between different formats.

Figure 40: Main UML Design.
The previous UML design shows in deeper detail the relations shown in the Nested package diagram.

The following UML diagrams have been used to show the methods included in the most important classes and the relations the classes present.

Figure 41: Bootstrap Server UML.

Figure 42: PeerUML.

Figure 43 Tracker UML.
Figure 44: UML Operations.

Figure 45: Application initial UML.
7.2 Software used

EclipseUML to program the implementation.
OMODO, for the UML design.
Windows Vista.
Microsoft Office

6.5 Hardware used

Two Laptops:
-Acer Aspire 5735z, Intel(R) Pentium(R) Dual CPU T3200 @ 2.00Ghz, 3.00GB RAM.
-HP Compaq 6730s, Intel(R) Core (TM)2 Duo CPU T5870 @2.00Ghz, 3.00GB RAM.

Wifi router Zyxel 660 HW-61.

Laptops picture:

Figure 46: Laptops

7.4 Scenario

We have set a simple scenery to test our protocol, using two laptops running our program we have been able to set one machine as client and the other as Bootstrap Server and Tracker. Both applications Tracker an Bootstrap server are running on a same computer put o completely independent processes.

Connection via Wifi or straight cable

Figure 47: scenario
8.1 Environmental Impact

In this chapter we will proceed to discuss the impact our protocol may have on the environment.

It is easy to believe that the design of a protocol applied to the networking context will have very little impact on the environment. But when we think it over, have you ever been to a Data Center? In data centers a great number of servers, switches and routers are stored and the energy consumption may be much greater than we might initially estimate.

According to the Uptime Institute, the typical 3-year cost (operating expenses + amortized capital expenses) of powering and cooling servers is approximately 1.5 times the cost of the server hardware itself, with projections of up to 22 times by the year 2012. The reason for increased energy consumption in the data center is, simply put, an increased demand for computing power. Since the heat generated by computer equipment must be removed in order to avoid overheating, increased computer load density results in increased heat density, which becomes a challenge for the HVAC equipment design. This power and cooling challenge is expected to continue.

The following pie chart obtained from the U.S Department of Energy, Energy Efficiency and Renewable Energy [2], illustrates the distribution of energy consume in a datacenter:

Figure 48: Datacenter power conversion & distribution

We believe that by standardizing the ppsp protocol we will slightly contribute to reduce the global electricity consumption. In the actual context, each company must handle its own equipment because proprietary protocols are used. But when standardization is applied, a single data center can be used to serve the interests of different streaming companies.

At this point we recall a fact explained at the beginning of this paper, a 50% of the traffic generated on the internet corresponds to video media. If we manage to unify the hardware equipment which deals with peer 2 peer streaming traffic by making use of a standard protocol, we are reducing the amount of energy globally consumed.
If we only applied the thermodynamic law to this case, we could conclude that by unifying the ppsp traffic generated by different companies to single equipments the number of connection each machine will have to support will increase considerably, and with it the electricity consumed by the Switches and routers CPU. But on the other hand, the amount of energy destined for the refrigeration of the equipments will indeed be reduced.

When the hardware is divided by streaming companies, each switch has to be refrigerated with its own fan, which means a fan per machine. In the future a single switch should be used at a same point, so just one switch fan would be active and the consume of electricity would be reduced.

If we apply this idea to all the datacenters globally spread containing different equipment destined to forward ppsp traffic, the save of energy can be more than significant.

Considering that many big companies manage proprietary datacenters, whole facilities are destined to allocate equipment working for streaming channels.

If we standardize the protocol, a single facility shared by different companies will be able to allocate the shared equipment forwarding traffic for all of them. The biggest achievement in this aspect is that a single aired conditioning infrastructure is now necessary to refrigerate a single data center, where when companies had independent data centers, each one had to have its own one.

### 8.2 Economical Issues

The planning and cost estimation chapter develops the time and the cost estimation of the realized project.

#### 8.2.1 Planning

The planning has focused on the following tasks:

- Research. A previous research has been done on peer-to-peer streaming protocol, understanding the problem statement has been our first task to determine which aspects needed to be treated. Research in ALTO protocol and multiple description coding has helped to improve the qualities of PPSP.

- Design. The design of the application, its architecture and functionalities is done once the problems to fulfill have been defined on the previous point.

- Implementation. This point consists of developing the points defined on our design. A simple prototype is developed at first. This prototype must succeed in implementing the most basic functionalities of our protocol,
after new functionalities are added to improve and give complexity to our project.

- Testing. The application to test the designed PPSP has been tested from different points and every time a new improvement was introduced. This has been done in order to assure a that the final set integrates correctly all the improvements.

- Documentation. The project has been documented following the standards dictated by the university. This documents includes the tasks carried out, conclusion, and captures of our implementation.

- Project Tracking. Periodical revisions of the state of the project have been carried out during the development by the tutor in charge. The revisions have helped to guide the projects evolution.

The following table shows the time estimation in developing the project.

Table 1

<table>
<thead>
<tr>
<th>Phase</th>
<th>Task</th>
<th>Description</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Research</td>
<td>First contact</td>
<td>Search on the internet PPSP information and monitoring the Working group.</td>
<td>40</td>
</tr>
<tr>
<td>Previous Research</td>
<td>Commercial PPSP</td>
<td>Study how commercial and academic PPSP mechanisms work</td>
<td>65</td>
</tr>
<tr>
<td>Design</td>
<td>Features required</td>
<td>Defining the bases and structure of PPSP messages</td>
<td>50</td>
</tr>
<tr>
<td>Design</td>
<td>Program draft</td>
<td>Design of the simple application</td>
<td>110</td>
</tr>
<tr>
<td>Implementation/ testing</td>
<td>First prototype p2p</td>
<td>Simple p2p application to set as base</td>
<td>20</td>
</tr>
<tr>
<td>Implementation/ testing</td>
<td>Developing and testing</td>
<td>New functionalities and improvements are added and tested until the latest version</td>
<td>140</td>
</tr>
<tr>
<td>Implementation/ testing</td>
<td>Live test</td>
<td>Once the application is fully operational captures are taken to proof it.</td>
<td>5</td>
</tr>
<tr>
<td>Documentation</td>
<td>Write PPSP Paper</td>
<td>Paper containing the research information.</td>
<td>100</td>
</tr>
<tr>
<td>Documentation</td>
<td>Adaptation</td>
<td>Adapting the paper to the established format.</td>
<td>20</td>
</tr>
</tbody>
</table>
8.2.2 Cost Estimation

It is hard to give a cost estimation to our project, there are lots of variables involved with costs hard to define.

Most of the cost would be applied to the time the student must dedicate in developing the project. A total of 525 hours approximately have been applied in this project.

If we consider that the average salary for a student would be of 7 Euros per hour that would make a total of 3675 Euros. But the Student will not receive a salary for the time invested, so the economical cost obtained from this point can only be considered theoretical.

The material needed to program the protocol and testing it is not expensive, most of it may be programmed using free license programs such as Eclipse for java developers. Switching and routing equipment have been supplied by the university, as well as the Internet connection.

We have used Microsoft Office to write the projects memory which is a pay license program, but other free license options exist such as Open Office.

The computers used to program and test the protocol on, did not need very high performing hardware so we could use ones of personal use.

**Theoretical global cost:**

<table>
<thead>
<tr>
<th>Table 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Student salary</strong></td>
<td>3675 €</td>
</tr>
<tr>
<td><strong>Microsoft Office license</strong></td>
<td>379 €</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>4054 €</strong></td>
</tr>
</tbody>
</table>

8.3 Personal Conclusion

The conclusion obtained from developing this project is that streaming content in the web is growing at a very high rate; perhaps faster than the network itself may stand.

New development and research must be carried out to adapt the network to new requirements, better quality video, more and more clients every day, avoiding bottlenecks and distributing the content providers globally are just the beginning.
If we aim to widely spread video sourcing through networking systems, a long path must be walked.

We believe that peer to peer is the most efficient solution because it allows spreading the content rapidly, and without the need of extra dedicated equipment when a peak occurs, helps avoiding bottlenecks and it is adaptable to networking changes in physical distribution and flow rates. Standardizing the protocol in charge of handling, signaling and controlling the peer-to-peer data flows is essential.

We may contribute to this cause by designing the protocol itself and proposing new ideas, or as it has been our case, keep an updated recall from how the standardized protocol is evolving.

Designing a simple program, as we have done, which implements most of the protocol requirements may be very useful as a reference for new developers programming for streaming media.

We believe that our project will help to gain a detailed view of what ppSP protocol is, and which role it plays in the evolution of video streaming, this information may be of great interest or for any one who would like to expand our research to other fields or any one who interested in designing new peer-2-peer video applications fulfilling the requirements that the new standard dictates.
CHAPTER 9. TERMINOLOGY AND CONCEPTS

Swarm: A swarm is a set of peers who are sharing the same live channel or VoD.

Chunk: A chunk is a basic unit of partitioned streaming, which is used by a peer for the purpose of storage, advertisement and exchange among peers [Sigcomm:P2P streaming].

Content Distribution Network (CDN) node: A CDN node refers to a network entity that usually is deployed at the network edge to store content provided by the original servers, and serves content to the clients located nearby topologically.

Data center (or datacentre): is a facility used to house computer systems and associated components, such as telecommunications and storage systems. It generally includes redundant or backup power supplies, redundant data communications connections, environmental controls (e.g., air conditioning, fire suppression) and security devices.

Live streaming: The scenario where all clients receive streaming content for the same ongoing event. The lags between the play points of the clients and that of the streaming source are small.

P2P cache: A P2P cache refers to a network entity that caches P2P traffic in the network, and either transparently or explicitly as a peer distributes content to other peers.

P2P streaming protocols: P2P streaming protocols refer to multiple protocols such as streaming control, resource discovery, streaming data transport, etc. which are needed to build a P2P streaming system.

Peer/PPSP peer: A peer/PPSP peer refers to a participant in a P2P streaming system. The participant not only receives streaming content, but also stores and uploads streaming content to other participants.

PPSP: PPSP refer to the key signaling protocols among various P2P streaming system components including the tracker and peer. PPSP are a part of P2P streaming protocols.

Tracker/PPSP tracker: A tracker/PPSP tracker refers to a directory service which maintains lists of peers/PPSP peers storing chunks for a specific channel or streaming file and answers queries from peers/PPSP peers for peer lists.

Video-on-demand (VoD): The scenario where different clients watch different parts of the media recorded and stored during past events.

Join Time: Join time is the absolute time when a peer registers on a Tracker. This value is recorded by the Tracker and is used to calculate Online Time.
Online Time: Online Time shows how long the peer has been in the P2P streaming system since it joins. This value indicates the stability of a peer, and it is calculated by Tracker when necessary.

Absolute Time: Absolute time is expressed as ISO 8601 [ISO.8601.2000] timestamps, using UTC (GMT). Fractions of a second may be indicated.

STUN: Simple Traversal of UDP over NATs.

TURN: Traversal Using Relay NAT

TLS: Transport Layer Security
CHAPTER 10. BIBLIOGRAPHY

Web page references:


Books, memories and drafts.


ANNEX 1. APLICATION TUTORIAL

A.1 User Messages Captures

The following captures represent the human interaction with our program:

Setting Bootstrap Server:

![Figure 49: Setting Bootstrap Server.](image-url)
Peer configuration:

Figure 50: Peer configuration.

Bootstrap server accepts the new peer:

Figure 51: Bootstrap accepts a new peer.
**Peer is set as Trucker:**

![Figure 52: Setting peer as Tracker.](image)

**Bootstrap server accepts to publish the channel:**

![Figure 53: Bootstrap accepts to publish the trackers channel](image)

**Tracker has published his channel:**

![Figure 54: Channel published](image)
Setting up a peer as client:

Figure 55: Setting peer as client.
Client has been accepted, partner selection:

Figure 56: Peer accepted by tracker

Figure 57: Partner selection

Peer accepted client as partner:

Figure 58: Accepting new partner.
On Both Peers it appears:

![Figure 59: New partner](image1)

Chunks exchange:

![Figure 60: Chunks exchange](image2)

Peer Leaves the Channel:

![Figure 61: Peer leaves channel](image3)
Tracker Receives:

Figure 62: Message received by tracker when peer leaves channel

A.2. System Out Captures

Print out captures from each node console:

A.2.1 Bootstrap Server

System.out:

*Setting computer as bootstrap Server and accepting peer connection.*

BS

---------->Setting computer as Bootstrap Server!
-----The BS port is: 9090
-----BS says: Client accepted

##### The BS has received: 1.0, 1, 5, 0, .., jaume&_.
body: jaume
Msg Parser_The message contains: [1.0, 1, 5, 0, .., jaume]

##### smethod = 1

##### subsreq received!!
jaume
the file will be named: peers
File already exists
body length = 2
wrfile jaume
wrfile /192.168.1.37
file updated
Out message done
string msg= 1.0, 2, 5, 0, .., 0&

********* The BS is sending: 1.0, 2, 5, 0, .., 0&
********* The BS is sending: [B@111a3ac
len=19

Peer send publish channel request and bootstrap servers allows to publish channel named ppsp.

-----BS says: Client accepted

##### The BS has received: 1.0, 3, L, 0, .., ppsp&_.
body: ppsp
Msg Parser_The message contains: [1.0, 3, L, 0, .., ppsp]

##### smethod = 3
///// Publ channel Req received!!!
Body: ppsp
Source IP: /192.168.1.37

the file will be named:Channels
File already exists
body length = 2
wrfile ppsp
wrfile /192.168.1.37
file updated
string msg= 1.0, 4, L, 0, .., 0&
********** The BS is sending: 1.0, 4, L, 0, .., 0&
********** The BS is sending: [B@13caecd
lenght: 19

New peer is accepted by the bootstrap Server.

-----BS says: Client accepted
############### The BS has received: 1.0, 1, 7, 0, .., natalia&
body: natalia
Msg Parser_The message contains: [1.0, 1, 7, 0, .., natalia]

##### smethod = 1
############### subsreq received!!
natalia
the file will be named:peers
File already exists
body length = 2
wrfile natalia
wrfile /192.168.1.154
file updated
Out message done
string msg= 1.0, 2, 7, 0, .., 0&
********** The BS is sending: 1.0, 2, 7, 0, .., 0&
********** The BS is sending: [B@1f42b49
lenght: 19

Bootstrap Server gets a channel list request containing the TAG science, and responds with the channel list message.

-----BS says: Client accepted
############### The BS has received: 1.0, 5, L, 0, .., science&
body: science
Msg Parser_The message contains: [1.0, 5, L, 0, .., science]

##### smethod = 5
///// Channel list Req received!!!
[B@87a5cc
Text representation of file channels:
EPSC/127.0.0.1
micasatv/127.0.0.1
house/127.0.0.1
formentera/127.0.0.1
gossip/192.168.1.37
saposverdes/192.168.1.37
gol/192.168.1.37
sgd/192.168.1.37
oki doki/192.168.1.37
ppsp/192.168.1.37
A.2.2 Tracker

System.out:

Establish computer as peer and login to the bootstrap server.

P

-------->Setting computer as Peer!
----->Peer Server is UP
  converting IP (string) to IP InetAddress....
  IP: laptop
  conversion completed

Converting 1.0, 1, 5, 0, .., jaume into message vector

body: jaume
  Msg Parser. The message contains: [1.0, 1, 5, 0, .., jaume]

string msg= 1.0, 1, 5, 0, .., jaume&
  length: 23

********** PeerClient is sending: 1.0, 1, 5, 0, .., jaume&
********** PeerClient is sending:[B@1503a3
  Message sent
Peer accepted by bootstrap Server and configured as trucker.

new connection

############ PeerServer has received: [B@1fae3c6 from: /192.168.1.154
############ Byte msg converted to string 1.0, 2, 5, 0, .. 0&_
Closing socket...

body: 0
Msg Parser_.The message contains: [1.0, 2, 5, 0, .. 0]

Message Being Handdled
##### method= 2
***** Bootstrap server response to subscription request:0
-----> Setting peer as tracker
Converting 1.0, 3, L, 0, .. ppsp into message vecto
body: ppsp
Msg Parser_.The message contains: [1.0, 3, L, 0, .. ppsp]

/192.168.1.154
string msg= 1.0, 3, L, 0, .. ppsp&
 lenght: 22
******* PeerClient is sending:1.0, 3, L, 0, .. ppsp&
******* PeerClient is sending:[B@5e3974
Message sent

-----> Set as traker
done
******* Message has been handdled, returning: [1.0, 2, 5, 0, .. 0]

Bootstrap server accepts to publish the trackers channel.

new connection

############ PeerServer has received: [B@1fae3c6 from: /192.168.1.154
############ Byte msg converted to string 1.0, 4, L, 0, .. 0&_
Closing socket...

body: 0
Msg Parser_.The message contains: [1.0, 4, L, 0, .. 0]

Message Being Handdled
##### method= 4
***** Bootstrap server response to Publish channel Request:0
->channel has been published

******* Message has been handdled, returning: [1.0, 4, L, 0, .. 0]

A new peer wants to join the Swamp (channel) and it is accepted, list of peer candidates (The Tracker itself included) is send.
* the Trackers IP is included because there a few candidates.

new connection

############ PeerServer has received: [B@1fae3c6 from: /192.168.1.154
############ Byte msg converted to string 1.0, 65, 2, 0, .. natalia.Alto Parameters.ADSL, LAPTOP, BARCELONA&
Closing socket...

body: natalia.Alto Parameters.ADSL
Msg Parser_ The message contains: [1.0, 65, 2, 0, ., natalia.Alto Parameters.ADSL]

Message Being Handled

### method= 65

########## Join Request received!!!
### body: natalia.Alto Parameters.ADSL

[B@13a317a

***** Text representation of file Traker-Members: Tracker/192.168.1.154
jaume.Alto Parameters.bcn/192.168.1.154
naty.Alto Parameters.adsl/192.168.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249

Out message done
[1.0, 66, 2, 0, ., Tracker/192.168.1.154
jaume.Alto Parameters.bcn/192.168.1.154
naty.Alto Parameters.adsl/192.168.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249
]
the file will be named:Traker-Clients
File already exists
body length = 2
wrfile natalia.Alto Parameters.ADSL
wrfile /192.168.1.154
file updated
string msg= 1.0, 66, 2, 0, ., Tracker/192.168.1.154
jaume.Alto Parameters.bcn/192.168.1.154
naty.Alto Parameters.adsl/192.168.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249
&
length: 251
********** PeerClient is sending:1.0, 66, 2, 0, ., Tracker/192.168.1.154
jaume.Alto Parameters.bcn/192.168.1.154
naty.Alto Parameters.adsl/192.168.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249
&
********** PeerClient is sending:[B@186768e
Message sent

********** Message has been handdled, returning: [1.0, 66, 2, 0, ., Tracker/192.168.1.154
jaume.Alto Parameters.bcn/192.168.1.154
naty.Alto Parameters.adsl/192.168.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249
]

Keepalive messeges are received from the accepted client periodically.

new connection

################ PeerServer has received: [B@1fae3c6from: /192.168.1.154
Tracker/Peer receives a Partnership request, and accepts to establish a partnership relation.

new connection

Tracker/Peer receives a Partnership request, and accepts to establish a partnership relation.

Buffermap exchange will begin shortly

Once a partnership relation is established, bufferMap exchange and peerList exchange (gossip messages) between peers takes place periodically.

*****Printing BufferMap:
Matrix length = 6
[1 1 1 1 1 1]
[0 1 1 1 1 1]
[1 1 0 1 1 0]
[0 1 1 0 0 0]
[1 0 1 0 1 1]
[0 0 0 1 0 1]
string msg = 1.0, 131, 6, 0, .., [1 1 1 1 1 1]
[ 0 1 1 1 1 1 ]
[ 1 1 0 1 1 0 ]
[ 0 1 1 0 0 0 ]
[ 1 0 1 0 1 1 ]
[ 0 0 0 1 0 1 ]
&
file updated
string msg = 1.0, 131, 6, 0, .., [1 1 1 1 1 1]
[ 0 1 1 1 1 1 ]
[ 1 1 0 1 1 0 ]
[ 0 1 1 0 0 0 ]
[ 1 0 1 0 1 1 ]
[ 0 0 0 1 0 1 ]
&
getMessagein is sending: [1.0, 131, 6, 0, .., [1 1 1 1 1 1]]
[ 0 1 1 1 1 1 ]
[ 1 1 0 1 1 0 ]
[ 0 1 1 0 0 0 ]
[ 1 0 1 0 1 1 ]
[ 0 0 0 1 0 1 ]

@ [1.0, 131, 6, 0, .., [1 1 1 1 1 1]]
[ 0 1 1 1 1 1 ]
[ 1 1 0 1 1 0 ]
[ 0 1 1 0 0 0 ]
[ 1 0 1 0 1 1 ]
[ 0 0 0 1 0 1 ]

B@4ae52

*****Partners: naty/192.168.1.37
cafetera/192.168.1.37
sabina/192.168.1.37
caro/192.168.1.37
jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
natalia/192.168.1.154
jaume/192.168.1.154

string msg = 1.0, 135, 251, 0, .., jaume[PList]: naty/192.168.1.37
cafetera/192.168.1.37
sabina/192.168.1.37
caro/192.168.1.37
jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
natalia/192.168.1.154
jaume/192.168.1.154
&
length: 116

********** PeerClient is sending: [1.0, 131, 6, 0, .., [1 1 1 1 1 1]]
[ 0 1 1 1 1 1 ]
[ 1 1 0 1 1 0 ]
Tracker/peer will receive buffer maps from his partners, compare them to his own and request the missing chunks.

Tracker/peer will receive buffer maps from his partners, compare them to his own and request the missing chunks.

new connection

new connection

new connection
Peer to Peer Streaming Protocol Standardization

[1 1 1 0 0 0 ]
[1 0 1 1 1 0] &

Closing socket...

body: [1 1 1 0 1 1 ]
[1 0 1 0 0 0 ]
[0 0 0 0 0 0 ]
[1 1 1 0 1 1 ]
[1 1 1 0 0 0 ]
[1 0 1 1 1 0]

Msg Parser. The message contains: [1.0, 131, 6, 0, .., [1 1 1 0 1 1 ]
[1 0 1 0 0 0 ]
[0 0 0 0 0 0 ]
[1 1 1 0 1 1 ]
[1 1 1 0 0 0 ]
[1 0 1 1 1 0 ]

Message Being Handled
##### method= 131
##### New BufferMap received from: /192.168.1.154
Body/buffermap:
[1 1 1 0 1 1 ]
[1 0 1 0 0 0 ]
[0 0 0 0 0 0 ]
[1 1 1 0 1 1 ]
[1 1 1 0 0 0 ]
[1 0 1 1 1 0 ]

***** Requesting chunks: [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]

string msg= 1.0, 132, 62, 0, .., [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]&
length: 83
********** PeerClient is sending:1.0, 132, 62, 0, .., [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]&

********** PeerClient is sending:[B@e1d5ea
Message sent

********** Message has been handdled, returning: [1.0, 132, 62, 0, .., [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]]

Tracker/peer receives chunksRequest from his partners and responds with the ones he has.

new connection
############## PeerServer has received: [B@1fae3c6from: /192.168.1.154
##############Byte msg converted to string1.0, 132, 62, 0, .., [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]&
Closing socket...

body: [0][2]
Msg Parser. The message contains: [1.0, 132, 62, 0, .., [0][2]]

Message Being Handled
##### method= 132
############## ChunkRequest received!!!!!! /192.168.1.154
##############Requested Chunks received: [0][2]
string msg= 1.0, 133, 33, 0, , jaume /[missing chunks] :
[0][2]&
length: 54
********** PeerClient is sending: 1.0, 133, 33, 0, , jaume /[missing chunks] :
[0][2]&
********** PeerClient is sending:[B@13adc56
Message sent
********** Message has been handdled, returning: [1.0, 133, 33, 0, , jaume /[missing chunks] :
[0][2]]

Tracker/peer receives chunks from his partners.

new connection
########## PeerServer has received: [B@291afffrom:/192.168.1.154
########## Byte msg converted to string1.0, 133, 31, 0, , natalia /[missing chunks] :
[0][2]& 1 1 0 1 0 0
[ 0 1 0 1 1 0 ]
[ 1 0 1 0 0 ]
[ 1 0 0 1 0 ]
& jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
jaume/192.168.1.37
&
Closing socket...

body: natalia /[missing chunks] :
[0][2]
Msg Parser_The message contains: [1.0, 133, 31, 0, , natalia /[missing chunks] :
[0][2]]

Message Being Handdled
##### method= 133
##### Missings Chunks obtained from:/192.168.1.154
natalia /[missing chunks] :
[0][2]

********** Message has been handdled, returning: [1.0, 133, 31, 0, , natalia /[missing chunks] :
[0][2]]

A.2.3 Peer Client

System.out:

Establish computer as peer and login to the bootstrap Server.

P
---------->Setting computer as Peer!
----->Peer Server is UP
converting IP (string) to IP InetAddress....
IP: 192.168.1.154
conversion compleated

Converting 1.0, 1, 7, 0, , natalia into message vector
body: natalia
Msg Parser_The message contains: [1.0, 1, 7, 0, , natalia]

string msg= 1.0, 1, 7, 0, , natalia &
length: 25
******** PeerClient is sending:1.0, 1, 7, 0, , natalia &
******** PeerClient is sending:[B@111a3ac
Message sent

Peer accepted by the bootstrap Server and sets himself as client. Next the client requests the channel list (channel tags -science- and location -SPAIN- are defined at this point).

new connection
################ PeerServer has received: [B@a83b8afrom:/192.168.1.154
################ Byte msg converted to string1.0, 2, 7, 0, , 0 &
Closing socket...

body: 0
Msg Parser_The message contains: [1.0, 2, 7, 0, , 0]

Message Being Handled
##### method= 2
*****Bootstrap server response to subscription request:0
-----> peer set as Client
-----> Setting peer as Client
*****List Channel request message is being created...
*****Channel list Req String is:1.0, 5, L, 0, , science, SPAIN, natalia
body: science
Msg Parser_The message contains: [1.0, 5, L, 0, , science]

Channel List Req Message has been parsed [1.0, 5, L, 0, , science]
string msg= 1.0, 5, L, 0, , science &
length: 25
******** PeerClient is sending:1.0, 5, L, 0, , science &
******** PeerClient is sending:[B@2808b3
Message sent

done client
******** Message has been handdled, returning: [1.0, 2, 7, 0, , 0]

Peer receives channel list from the Bootstrap Server.

new connection
################ PeerServer has received: [B@a83b8afrom:/192.168.1.154
################ Byte msg converted to string1.0, 6, L, 0, , EPSC/127.0.0.1
micasatv/127.0.0.1
house/127.0.0.1
formentera/127.0.0.1
gossip/192.168.1.37
saposverdes/192.168.1.37
gol/192.168.1.37
sdg/192.168.1.37
oki doki/192.168.1.37
ppsp/192.168.1.37
&_
Closing socket...
body: EPSC/127.0.0.1
micasatv/127.0.0.1
house/127.0.0.1
formentera/127.0.0.1
gossip/192.168.1.37
saposverdes/192.168.1.37
gru/192.168.1.37
sdg/192.168.1.37
oki doki/192.168.1.37
ppsp/192.168.1.37

Msg Parser. The message contains: [1.0, 6, L, 0, ., EPSC/127.0.0.1
micasatv/127.0.0.1
house/127.0.0.1
formentera/127.0.0.1
gossip/192.168.1.37
saposverdes/192.168.1.37
gru/192.168.1.37
sdg/192.168.1.37
oki doki/192.168.1.37
ppsp/192.168.1.37
]

Message Being Handdled
##### method= 6
##### Channels included in body: EPSC/127.0.0.1
micasatv/127.0.0.1
house/127.0.0.1
formentera/127.0.0.1
gossip/192.168.1.37
saposverdes/192.168.1.37
gru/192.168.1.37
sdg/192.168.1.37
oki doki/192.168.1.37
ppsp/192.168.1.37

A channel is selected by the client and request is sent to the relevant Tracker with the clients ALTO parameters.

*****You have selected the following Tracker[@IP] 192.168.1.37

IP: jaume
***** join Request body: natalia, ADSL, LAPTOP, BARCELONA
********** The elements of JoinRequest: [1.0, 65, 2, 0, ., natalia.Alto Parameters.ADSL, 
LAPTOP, BARCELONA]

string msg= 1.0, 65, 2, 0, ., natalia.Alto Parameters.ADSL, 
LAPTOP, BARCELONA&
lenght: 66
********** PeerClient is sending:1.0, 65, 2, 0, ., natalia.Alto Parameters.ADSL, 
LAPTOP, BARCELONA&
********** PeerClient is sending:B@1e152c5
Message sent

********** Message has been handdled, returning: [1.0, 65, 2, 0, ., natalia.Alto Parameters.ADSL, 
LAPTOP, BARCELONA]

The Tracker accepts the peer to join his channel and responds with a list of possible peers-partners.
new connection

PeerServer has received: [B@a83b8afrom:/192.168.1.37

PeerServer has received: 1.0, 66, 2, 0,, 0:Tracker/192.168.1.154

jaume.Alto Parameters.bcn/192.168.1.154
naty.Alto Parameters.adsl/192.168.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249

Closing socket...

body: 0:Tracker/192.168.1.154
jaume.Alto Parameters.bcn/192.168.1.154
naty.Alto Parameters.adsl/192.168.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249

Msg Parser_ The message contains: [1.0, 66, 2, 0, ,, 0:Tracker/192.168.1.154
jaume.Alto Parameters.bcn/192.168.1.154
naty.Alto Parameters.adsl/192.168.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249

Message Being Handdled

##### method= 66
*****Tracker Response= 0
Partners list: 0:Tracker/192.168.1.154
jaume.Alto Parameters.bcn/192.168.1.154
naty.Alto Parameters.adsl/192.168.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249

You have been accepted by the tracker!

Client will periodically send keep alive messages including his id to the trucker.

string msg= 1.0, 68, 7, 0, , natalia&
lenght: 26
********** PeerClient is sending:1.0, 68, 7, 0, , natalia&
********** PeerClient is sending:[B@84da23
Message sent

Client selects the Partners IP with which he would like to establish a partnership relation and Exchange chunks.

partner candidates: 192.168.1.37, 10.183.220.249
entered 2 addresses

**********getMessagein is sending: [1.0, 66, 2, 0,, 0:Tracker/192.168.1.154
jaume.Alto Parameters.bcn/192.168.1.154
naty.Alto Parameters.adsl/192.168.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249
]

string msg= 1.0, 68, 7, 0, .., natalia&
length: 26
********** PeerClient is sending:1.0, 68, 7, 0, .., natalia&
********** PeerClient is sending:[B@5f8172
Message sent

IP: jaume
IP: 10.183.220.249
2Candidates have been selected

PartReq-> i =2
ips = [Ljava.net.InetAddress;@1df8b99
[Ljava.net.InetAddress;@1df8b99
PartReq->2 partner candidates have been selected.
*****Partnership Request message includes:
natalia
ADSL, Barcelona
natalia, ADSL, Barcelona

**********The elements of Partnership Request: [1.0, 129, 2, 0, .., natalia, ALTO: ADSL, Barcelona]

*****PartReq-> Outmsg: [1.0, 129, 2, 0, .., natalia, ALTO: ADSL, Barcelona]
string msg= 1.0, 129, 2, 0, .., natalia, ALTO: ADSL, Barcelona&
length: 50
********** PeerClient is sending:1.0, 129, 2, 0, .., natalia, ALTO: ADSL, Barcelona&
********** PeerClient is sending:[B@176e552
Message sent

PartReq-> 0

**Partnership has been accepted.**

new connection
########## PeerServer has received: [B@93dcdfrom:/192.168.1.37
##########Byte msg converted to string1.0, 130, 2, 0, .., jaume/&/192.168.1.154
jaume.Alto Parameters.bcn/192.168.1.154
naty.Alto Parameters.adsl/192.168.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249

Closing socket...

body: 0/ jaume
Msg Parser_The message contains: [1.0, 130, 2, 0, .., 0/ jaume]

Message Being Handdled
##### method= 130
------>New partnership relation stablished!!!!
string msg= 1.0, 68, 3, 0, .., natalia&
length: 22
********** PeerClient is sending:1.0, 68, 3, 0, .., natalia&
********** PeerClient is sending:[B@ccc588
Message sent

Buffermap exchange will begin shortly
the file will be named: partners
getMessagein is sending: [1.0, 130, 2, 0, .., 0/ jaume]

File already exists
*****Printing BufferMap:
body length = 2
wrfile jaume
wrfile /192.168.1.37
Matrix length = 6
[[1 1 1 0 1 1],
 [1 0 1 0 0 0],
 [0 0 0 0 0 0],
 [1 1 1 0 1 1],
 [1 1 1 0 0 0],
 [1 0 1 1 1 0]]

string msg= 1.0, 131, 6, 0, .., [1 1 1 0 1 1]
[[1 0 1 0 0 0],
 [0 0 0 0 0 0],
 [1 1 1 0 1 1],
 [1 1 1 0 0 0],
 [1 0 1 1 1 0]]

file updated
********** Message has been handled, returning: [1.0, 131, 6, 0, .., [1 1 1 0 1 1]]
[[1 0 1 0 0 0],
 [0 0 0 0 0 0],
 [1 1 1 0 1 1],
 [1 1 1 0 0 0],
 [1 0 1 1 1 0]]

When the partnership request is accepted, the client begins to receive BufferMap messages, and peer List messages periodically.

new connection
########### PeerServer has received: [B@a83b8a from: /192.168.1.37
###########Byte msg converted to string1.0, 131, 6, 0, .., [1 1 1 1 1 1]
[0 1 1 1 1 1]
[1 1 0 1 1 0]
[0 1 1 0 0 0]
[1 0 1 0 1 1]
[0 0 0 0 0 0]
[0 0 0 1 0 1]

&8.1.37
cafetera.Alto Parameters.adsl/192.168.1.37
caro.Alto Parameters.adsl/192.168.1.37
jaume.Alto Parameters.bac/10.183.220.249

Closing socket...

body: [1 1 1 1 1 1]
[0 1 1 1 1 1]
[1 1 0 1 1 0]
[0 1 1 0 0 0]
[1 0 1 0 1 1]
[0 0 0 1 0 1]

Msg Parser_ The message contains: [1.0, 131, 6, 0, .., [1 1 1 1 1 1]
[0 1 1 1 1 1]
[1 1 0 1 1 0]
Client/Peer compares the received bufferMap with his own bufferMap and requests to his partner the Missing chunks.

***** Requesting chunks: [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]

string msg= 1.0, 132, 62, 0, .., [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]&
length: 83

********** PeerClient is sending: 1.0, 132, 62, 0, .., [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]&
********** PeerClient is sending: [B@b6e39f
Message sent

********** Message has been handled, returning: [1.0, 132, 62, 0, .., [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]]

Peer receives PartnerList message (Gossip Message).

new connection

******* PeerServer has received: [B@a83b8af from: /192.168.1.37
******* Byte msg converted to string: 1.0, 135, 251, 0, .., jaume[PL]: naty/192.168.1.37
cafetera/192.168.1.37
sabina/192.168.1.37
caro/192.168.1.37
jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
natalia/192.168.1.154

Closing socket...

body: jaume[PL]: naty/192.168.1.37
cafetera/192.168.1.37
sabina/192.168.1.37
caro/192.168.1.37
jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
Peer to Peer Streaming Protocol Standardization

natalia/192.168.1.154

Msg Parser: The message contains: [1.0, 135, 251, 0, , jaume[PL]: naty/192.168.1.37
cafetera/192.168.1.37
sabina/192.168.1.37
caro/192.168.1.37
jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
natalia/192.168.1.154
]

Message Being Handled
##### method= 135
############### new PeerDiscoveryMessage received form: /192.168.1.37!!!
Partners obtained:
jaume[PL]: naty/192.168.1.37
cafetera/192.168.1.37
sabina/192.168.1.37
caro/192.168.1.37
jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
natalia/192.168.1.154
]

********** Message has been handled, returning: [1.0, 135, 251, 0, , jaume[PL]:
naty/192.168.1.37
cafetera/192.168.1.37
sabina/192.168.1.37
caro/192.168.1.37
jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
natalia/192.168.1.154
]

Peer receives from partner the Messing chunks and partner List.

new connection
################ PeerServer has received: [B@a83b8afrom: /192.168.1.37
################ Byte msg converted to string: 1.0, 133, 33, 0, , jaume /[missing chunks] :
[0][2]& 0 0 0 0 1 ]
[ 1 0 1 0 0 1 ]
[ 0 1 1 0 1 0 ]
[ 0 0 0 0 1 1 ]
&
jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
natalia/192.168.1.154

Closing socket...
body: jaume /[missing chunks]: [0][2]

Msg Parser _The message contains: [1.0, 133, 33, 0, .. jaume /[missing chunks]: [0][2]]

Message Being Handdled

##### method= 133
##### Missings Chunks obtained from: /192.168.1.37
jaume /[missing chunks]: [0][2]

********** Message has been handdled, returning: [1.0, 133, 33, 0, .. jaume /[missing chunks]: [0][2]]

_Peer receives Chunk requests and he responds with the ones he has._

new connection

########## PeerServer has received: [B@93dcdfrom: /192.168.1.37
########## Byte msg converted to string 1.0, 132, 62, 0, .. [0][2], [1][1], [2][5], [3][1], [3][3], [4][5],
[5][0], [5][4][ 1 0 1 0 1 0 ]
&
jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
natalia/192.168.1.154

Closing socket...

body: [0][2]

Msg Parser _The message contains: [1.0, 132, 62, 0, .. [0][2]]

Message Being Handdled

##### method= 132

########## ChunkRequest received!!!!! /192.168.1.37

########## Requested Chunks received: [0][2]

string msg= 1.0, 133, 31, 0, .. natalia /[missing chunks]: [0][2]&
lenght: 52

********** PeerClient is sending:1.0, 133, 31, 0, .. natalia /[missing chunks]: [0][2]&
********** PeerClient is sending:[B@e28b9
Message sent

********** Message has been handdled, returning: [1.0, 133, 31, 0, .. natalia /[missing chunks]: [0][2]]

This process will continue until the peer leaves the system.

..........

new connection

########## PeerServer has received: [B@a83b8af from: /192.168.1.37
########## Byte msg converted to string 1.0, 133, 33, 0, .. jaume /[missing chunks]: [0][0]& 0 1 0 0 0 0 ]
[ 1 0 1 0 1 1 ]
[ 1 1 0 0 1 1 ]
Closing socket...

body: jaume /[missing chunks]: [0][0]
Msg Parser_ The message contains: [1.0, 133, 33, 0, .., jaume /[missing chunks]: [0][0]]

Message Being Handled
####### method= 133
##### Missings Chunks obtained from: /192.168.1.37
jaume /[missing chunks]: [0][0]

********** Message has been handdled, returning: [1.0, 133, 33, 0, .., jaume /[missing chunks]: [0][0]]

new connection
############### PeerServer has received: [B@a83b8afrom: /192.168.1.37
############### Byte msg converted to string1.0, 131, 6, 0, .., [ 0 0 0 1 0 0 ]
[ 0 0 0 0 0 ]
[ 0 0 0 1 0 1 ]
[ 1 1 0 0 1 0 ]
[ 1 0 1 1 0 1 ]
[ 1 1 0 1 0 1 ]
&
jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
natalia/192.168.1.154

Closing socket...

body: [ 0 0 0 1 0 0 ]
[ 0 0 0 0 0 ]
[ 0 0 0 1 0 1 ]
[ 1 1 0 0 1 0 ]
[ 1 0 1 1 0 1 ]
[ 1 1 0 1 0 1 ]

Msg Parser_ The message contains: [1.0, 131, 6, 0, .., [ 0 0 0 1 0 0 ]
[ 0 0 0 0 0 ]
[ 0 0 0 1 0 1 ]
[ 1 1 0 0 1 0 ]
[ 1 0 1 1 0 1 ]
[ 1 1 0 1 0 1 ]
]

Message Being Handdled
####### method= 131
New BufferMap received from: /192.168.1.37

Body/buffermap:

```
[ 0 0 0 1 0 0 ]
[ 0 0 0 0 0 0 ]
[ 0 0 0 1 0 1 ]
[ 1 1 0 0 1 0 ]
[ 1 0 1 1 0 1 ]
[ 1 1 0 1 0 1 ]
```

Requesting chunks: [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]

```
string msg= 1.0, 132, 62, 0, .., [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]&
length: 83
```

PeerClient is sending: 1.0, 132, 62, 0, .., [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0], [5][4]&

PeerClient is sending: [B@e61fd1
Message sent

Message has been handled, returning: [1.0, 132, 62, 0, .., [0][2], [1][1], [2][5], [3][1], [3][3], [4][5], [5][0],

new connection

PeerServer has received: [B@a83b8af from: /192.168.1.37
Byte msg converted to string: 1.0, 133, 33, 0, .., jaume / [missing chunks] :

```
[0][2]& 0 0 1 0 1 ]
[ 1 1 0 1 0 ]
[ 1 0 1 1 0 ]
[ 1 1 0 1 0 ]
```

jaume/192.168.1.37
sabina/192.168.1.37
gol/192.168.1.37
jaume/10.183.220.249
alberto/10.183.220.249
natalia/192.168.1.154

Closing socket...

body: jaume / [missing chunks] :

```
[0][2]
```

Msg Parser: The message contains: [1.0, 133, 33, 0, .., jaume / [missing chunks] :

```
[0][2]
```

Message Being Handdled

```
method= 133
```

Missings Chunks obtained from: /192.168.1.37
jaume / [missing chunks] :

```
[0][2]
```

Message has been handled, returning: [1.0, 133, 33, 0, .., jaume / [missing chunks] :

```
[0][2]
```

Message has been handled, returning: [1.0, 133, 33, 0, .., jaume / [missing chunks] :

```
[0][2]
```

getMessagein is sending: [1.0, 68, 7, 0, .., natalia

string msg= 1.0, 67, 7, 0, .., natalia&
string msg= 1.0, 67, 7, 0, .., natalia&
Error in PeerClient

********** Leaving the Channel!!!
Finally the peer leaves the channel.