laSalle

UNIVERSITAT RAMON LLULL

Escola Tècnica Superior d'Enginyeria La Salle

Treball Final de Màster

Màster Universitari en Enginyeria de Xarxes i Telecomunicació

Ampliació de l'estudi i implementació del protocol d'streaming peer to peer: PPSP

Alumne
Francisco Javier García López

Professor Ponent
Gabriel Fernàndez Ubiergo

ACTA DE L'EXAMEN
DEL TREBALL FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Francisco Javier García López

va exposar el seu Treball de Fi de Carrera, el qual va tractar sobre el tema següent:

Ampliació de l'estudi i implementació del protocol d'streaming peer to peer: PPSP

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Abstract

PPSP és l'abreviació de *Peer to Peer Streaming Protocol*. El seu objectiu és definir un protocol de localització i transmissió de dades en temps real de manera eficient a partir de múltiples fonts amb parts diferents en un entorn *peer-to-peer* en *streaming*. Dins d'aquest marc, l'objectiu principal d'aquest projecte és fer una ampliació de l'estudi del protocol, fet al Projecte Final de Carrera "*Estudi i implementació del nou protocol de Streaming Peer to Peer, PPSP*"; amb funcionalitats i requeriments que PPSP ha de complir per facilitar l'*streaming* P2P, així com amb l'estudi de l'*streaming* P2P per nodes mòbils. A més d'aquesta ampliació de l'estudi del protocol, s'ha desenvolupat una aplicació en *Android* per l'intercanvi de fluxos de missatges, compatible amb l'aplicació en C++, i així veure de forma pràctica amb la implementació del protocol com a *dissector* per *Wireshark*, el seu funcionament a través de la xarxa.

Abstract

PPSP es la abreviación de *Peer to Peer Streaming Protocol*. Su objetivo es definir un protocolo de localización y transmisión de datos en tiempo real de manera eficiente a partir de múltiples fuentes con partes diferentes en un entorno *peer-to-peer* en *streaming*. Dentro de este marco, el objetivo principal del proyecto es hacer una ampliación del estudio del protocolo, hecho en el Proyecto Final de Carrera "*Estudi i implementació del nou protocol de Streaming Peer to Peer, PPSP*"; con funcionalidades y requerimientos que PPSP debe cumplir para facilitar el *streaming* P2P, así como con el estudio del *streaming* P2P para nodos móviles. Además de esta ampliación del estudio del protocolo, se ha desarrollado una aplicación en *Android* para el intercambio de flujos de mensajes, compatible con la aplicación en C++, y así ver de forma práctica con la implementación del protocol como *dissector* para *Wireshark*, su funcionamiento a través de la red.

Abstract

PPSP is the abbreviation of *Peer to Peer Streaming Protocol*. Its aim is to define a protocol for locating and data transmission in real time efficiently from multiple sources with different parts in a *peer-to-peer streaming* environment. Within this framework, the project's main objective is making an extension of the study of this protocol, made in the Final Project "*Estudi i implementació del nou protocol de Streaming Peer to Peer, PPSP*", with features and requirements that PPSP must meet to facilitate P2P *streaming*, as well as the study of P2P streaming mobile nodes. In addition to this expansion of the protocol study, has been developed an application in Android that includes an exchange of message flows, compatible with the application in C++, to view in a practical way by the implementation of the protocol as dissector for *Wireshark*, its performance across the network.

Resum

Amb l'objectiu de voler estandarditzar els protocols de senyalització entre els diversos components d'un sistema P2P *streaming* (*Tracker* i *peers*) des de l'IETF es proposa i crea un conjunt de protocols anomenat PPSP (*Peer to Peer Streaming Protocol*), que són una part dels protocols d'*streaming* P2P, amb el propòsit de desenvolupar protocols estàndards de senyalització per múltiples tipus d'entitats. Aquest projecte, per tant, es situa dintre del marc de l'*streaming* P2P, tenint com a objectiu principal l'estudi del protocol PPSP, que és un protocol de localització i transmissió de dades en temps real de forma eficient, a partir de múltiples fonts amb parts diferents en un entorn *peer-to-peer*.

Actualment, hi ha múltiples solucions per *streaming* de P2P disponibles i, tot i que algunes d'elles ja s'han deixat de banda, la majoria encara continuen sota estudi. Aquest fet, suposa una dificultat extra a l'hora de voler integrar els sistemes P2P a la infraestructura d'entrega de contingut global degut a incompatibilitats. Aquest és un altre motiu per desenvolupar un estàndard, que haurà de complir uns requeriments.

Així doncs, primer es donarà una visió general del protocol, així com de les seves característiques i requeriments, assentant les bases del mateix, per parlar després del *PPSP Tracker Protocol* i el *PPSP Peer Protocol*, passant a parlar després de l'*streaming* P2P per nodes mòbils i entrar així a la part pràctica del projecte, el desenvolupament en *Android* d'una aplicació per l'enviament de missatges de senyalització usant el *PPSP Tracker Protocol*. A més, hi ha un estudi matemàtic de la cadència d'enviament dels missatges periòdics del protocol.

Actualment encara s'està definint el protocol i no està previst que fins Abril de l'any 2012 el grup de treball de l'IETF que treballa amb el desenvolupament del protocol PPSP es dissolgui si no ha arribat enlloc o es faci una nova definició del *charter* per tal de marcar noves metes, assumint que el protocol estarà llavors prou avançat.

Per a completar la part d'estudi del protocol PPSP, s'ha desenvolupat una aplicació en *Android*, compatible amb el node desenvolupat en C++, que permet veure l'intercanvi de missatges entre *peer* i *Tracker* de amb una implementació del *PPSP Tracker Protocol*, recolzada amb l'ús del *dissector* del protocol PPSP, que permet que *Wireshark* reconegui els paquets del protocol PPSP i els tracti com a tals.

El desenvolupament de l'aplicació que acompanya a l'ampliació de l'estudi del protocol, així com tot el projecte, s'ha realitzat com a projecte emmarcat dins del *Departament de Tecnologies Media* de *Transferència de Tecnologia La Salle*.

Índex

Índex de figures

Acrònims

AN: Acces Network

API: Application Programming Interface

CDN: Content Distribution Network

CN: Correspondent Node

DPI: Deep Packet Inspection

DoS: Denial of Service

GPS: *G*lobal Positioning System

IETF: Internet Engineering Task Force

IRTF: Internet Research Task Force

ISP: Internet Service Provider

HI: Host Identity

HIP: Host Identity Protocol

MIP: Mobile IP Protocol

NFC: Near Field Communication

OHA: Open Handset Alliance

P2P: Peer to peer

PMIP: Proxy Mobile IP Protocol

PPSP: Peer to Peer Streaming Protocol

RELOAD: REsource LOcation And Discovery Base Protocol

RFC: Request For Comments

RTT: Round-Trip delay Time

SGL: Scalable Graphics Library

SMS: Short Message Service

SSL: Secure Sockets Layer

VPN: Virtual Private Network

WG: Working Group

XML: eXtensible Markup Language

1. Introducció

A finals de la dècada dels 90, van aparèixer els primers programes de *peer to peer*, és a dir, els primers sistemes que servien com a mètode per l'intercanvi massiu d'arxius, mp3 principalment, tot i que abastava també altres tipus de fitxers.

Amb el temps, la tecnologia P2P es va anar estenent a altres àrees d'Internet, ja no només per l'intercanvi de fitxers com es venia fent, sinó per altres aplicacions de comunicació *one-to-one* com poden ser Veu sobre IP (VoIP) i missatgeria instantània o *one-to-many* com jocs *online*, la citada compartició d'arxius i l'*streaming*.

De fet, el tràfic P2P està creixent cada cop més i, actualment, suposa la major part del tràfic que circula per Internet. I és més, avui en dia, les aplicacions d'*streaming* de vídeo en concret són les aplicacions que més ample de banda demanen. A l'àrea d'*streaming* la popularitat del P2P en temps real ha crescut molt. Tot i això, encara que s'utilitzen de forma massiva aquest tipus d'aplicacions i que hi ha un interès cada cop més elevat, tant acadèmic com comercial, les aplicacions *streaming* P2P en temps real a Internet encara representen un tema obert. Això es deu sobre tot a que actualment hi ha diverses solucions propietàries no compatibles entre si, ja que cada desenvolupador ha anat adaptant la idea de l'*streaming* P2P segons les seves necessitats o el que vulgui oferir.

Degut a això, des del IETF es va proposar la creació d'un grup de treball que es dediqués a la investigació i el desenvolupament d'un protocol d'*streaming peer to peer* no propietari, i així cobrir les creixents necessitats que van sorgint en aquest camp; així com estandarditzar els protocols de control i transferència d'*streaming* per les aplicacions d'*streaming* P2P.

És dins d'aquest entorn on es veu emmarcat aquest projecte, treballant amb la base del desenvolupament d'aquest nou protocol d'*streaming peer to peer* no propietari, de forma que aquest acabi convertint-se en un protocol estàndard d'Internet, i fer una ampliació de l'estudi del mateix, que es va fer al Projecte Final de Carrera titulat "*Estudi i implementació del nou protocol de Streaming Peer to Peer, PPSP*", implementant una aplicació que complementi la part pràctica del mateix, i així poder veure el funcionament del protocol de forma pràctica.

En els diferents apartats que segueixen a continuació, s'aniran assentant les bases teòriques del PPSP, fent primerament una explicació de què és l'*streaming peer to peer*, juntament amb les seves característiques principals i requeriments, així com disseny de la seva arquitectura, així com els protocols que l'han de formar.

A aquests protocols, degut a la importància que tenen per si sols, se'ls hi dedica un apartat sencer a cadascun, sent el primer d'ells, el *PPSP Tracker Protocol*, el que té major rellevància en aquest projecte. Aquest protocol, és el que s'encarrega de les comunicacions entre *peer* i *Tracker* en una xarxa *streaming* P2P amb tot un seguit de missatges diferenciats.

D'altra banda, el segon protocol del que es parla és el *PPSP Peer Protocol*, que s'encarrega de les comunicacions entre els peers a la xarxa.

Una vegada feta l'introducció de PPSP i vist els requeriments, així com els protocols que el conformen, es dedica un apartat per parlar sobre l'*streaming* a nodes mòbils. El que es pretén amb aquest apartat es donar una visió general de l'integració dels dispositius mòbils a les xarxes d'*streaming* P2P, i els problemes que comporten aquests nodes respecte als *peers* fixes.

Per acabar, s'ha redactat un apartat per mostrar els resultats obtinguts a la part pràctica. Tot i això, abans d'entrar a veure'ls, es fa una introducció teòrica del sistema operatiu d'*Android*, es parla de la comunicació entre aplicacions desenvolupades en Java i C++, i es parla de les modificacions realitzades a l'aplicació desenvolupada anteriorment, per tal de poder compatibilitzar ambdues aplicacions. Un cop parlat d'aquests canvis, s'introdueix l'aplicació desenvolupada en Android i es veu detalladament el seu diagrama de classes i el seu funcionament.

A més, es veu un estudi matemàtic fet per determinar la cadència d'enviament dels missatges periòdics del protocol. Després d'aquest estudi, es mostren els resultats obtinguts amb el desenvolupament de l'aplicació en *Android* (juntament amb l'aplicació desenvolupada en C++), que permet simular una xarxa P2P que intercanviï els missatges del *PPSP Tracker Protocol*, gràcies a les eines desenvolupada on es troba la classe que implementa els missatges i al *dissector* programat en *Lua* per *Wireshark*, amb el que es poden capturar els missatges corresponents al protocol per poder-los veure correctament amb l'analitzador de xarxes.

Finalment, s'extreuen conclusions de l'estat actual del projecte i sobre les línies de futur existents del mateix, així com també es contemplen possibles evolucions de l'aplicació a curt i llarg termini.

2. Peer to Peer *Streaming* Protocol

2.1. Necessitat del protocol

Cada cop més, les aplicacions *streaming*[1] de vídeo atreuen més usuaris a usar aquestes aplicacions i el contingut *online* de vídeo. En un entorn de VoD, diferents usuaris veuen diferents parts del contingut de vídeo, així que un *peer* normalment manté un *buffer* per a compartir contingut de vídeo amb altres *peers*. D'altra banda, en un entorn de contingut en viu, degut a que tots els *peers* estan interessats en el que succeeix en l'instant actual, un dels possibles rols d'un *peer* és el de demanar contingut de vídeo de la font en viu i llavors transmetre aquest contingut cap a altres *peers*, reduint així el treball que ha de fer la font.

Les aplicacions d'*streaming* P2P[2] adopten una arquitectura d'*streaming* descentralitzada on el contingut mèdia es comparteix entre *peers* que a més de descarregar aquest contingut, comparteixen el contingut entre ells, pujant parts del mateix cap als altres. Entre els avantatges d'aquest sistema descentralitzat d'*streaming* s'inclou una menor carrega de treball als servidors d'*streaming*, cosa que es veu reflectida en un menor cost, i una millor escalabilitat d'*streaming* en un nombre gran d'usuaris. El problema és que la majoria de les aplicacions d'*streaming* P2P actuals utilitzen protocols propietaris, fet que provoca que sigui impossible per moltes altres aplicacions reutilitzar la totalitat o part dels components d'aquests protocols propietaris per implementar un sistema d'*streaming* descentralitzat. Dins d'aquest raonament surt la idea de definir des del IETF un estàndard de protocol obert per *streaming* P2P, que passa a conèixer-se com a PPSP, i que podria aportar grans beneficis a moltes aplicacions a través d'una arquitectura descentralitzada que permetrà costos d'infraestructura reduïts i millor escalabilitat.

Un possible escenari el trobem a dins d'una xarxa de distribució de contingut o *content distribution network (CDN)*, desplegada per proveïdors de continguts, on PPSP es pot fer servir per a reduir la càrrega d'*streaming* dels *edge servers* i millorar l'escalabilitat d'*streaming* compartint el contingut mèdia entre els diferents usuaris de la xarxa, així com amb els *edge servers*. El motiu de no fer server P2PSIP és que a P2P, la informació de contingut de cada *peer* és molt dinàmica i en temps real, el que significa que el simple manteniment d'aquest tipus d'informació a una xarxa P2PSIP pot causar sobrecarrega d'aquest. Per tant, a l'*streaming* P2P, és sempre millor mantenir la informació de contingut local als *peers* distribuïts i usar PPSP per a descobrir quin *peer* te cada contingut. La xarxa P2PSIP pot ser usada per reemplaçar el *Tracker* per implementar registre distribuït dels *peers*.

2.2. Introducció al protocol

Peer to Peer Streaming Protocol (PPSP) [3] és el nom amb el que es coneix als protocols de senyalització que s'apliquen al *Tracker* i als *peers* en un sistema d'*streaming peer to peer*. PPSP servirà com una tecnologia capacitadora, basant-se en les experiències de desenvolupament dels sistemes d'*streaming* P2P actuals. El seu disseny li permetrà la integració amb els esforços fets per l'IETF en localització de recursos distribuïts, localització de tràfic i mecanismes de control d'*streaming*. Permet, a més, integració efectiva amb infraestructures *edge* així com amb *caché* i equipament mòbil.

Així doncs, PPSP inclourà a dins de la seva definició el *PPSP Tracker Protocol*, que és un protocol de senyalització entre els *Trackers* i els *peers* de PPSP; i el *PPSP Peer Protocol*, un protocol també de senyalització però en aquest cas només entre *peers*. Ambdós protocols, als apartats 3 i 4 respectivament, es veuran amb tot detall.

Abans d'entrar a parlar sobre el disseny de l'arquitectura i l'abast del protocol als següents sub-apartats, s'introdueix lleugerament l'organisme que s'encarrega de normalitzar l'arquitectura i els protocols d'Internet (IETF), així com els documents amb els que es fan les propostes oficials per a desenvolupar nous protocols a Internet (RFC).

2.2.1. IETF

Internet Engineering Task Force (IETF)[4] és una organització internacional oberta de normalització que té com a objectius el contribuir a la enginyeria d'Internet, actuant a diverses àrees com: transport, enrutament i seguretat. Va ser creada als Estats Units l'any 1986 i és mundialment coneguda per ser l'entitat que regula les propostes i els estàndards d'Internet, coneguts com *Request For Comments* (RFC).

El IETF és una institució sense ànim de lucre i oberta a la participació de qualsevol persona, amb l'objectiu de velar per què l'arquitectura d'Internet i els protocols que la conformen funcionin correctament. Està considerada com l'organització amb major autoritat per establir modificacions dels paràmetres tècnics sota els que funciona la xarxa. No en va, la IETF es composa de tècnics i professionals a l'àrea de les xarxes, tals com investigadors, dissenyadors de xarxes, administradors, venedors, ... entre d'altres.

Donat que l'organització abasta diverses àrees diferents, s'utilitza una metodologia de divisió en grups de treball (*Working Groups* o WG), cadascun dels quals es forma per a treballar en un tema en particular, amb l'intenció de concentrar els esforços a cada part concreta per separat.

2.2.2. RFC

Els *Request For Comments* (RFC)[5] són una sèrie de notes sobre Internet que van començar a publicar-se l'any 1969. Cadascuna d'elles individualment és un document on el contingut el forma una proposta oficial per un nou protocol d'Internet, que s'explica amb tot detall per a què en cas de ser acceptat es pugui implementar sense cap mena d'ambigüitats.

Qualsevol persona pot enviar una proposta de RFC a l'organisme que forma la IETF, però es aquest organisme qui decidirà finalment si el document es converteix oficialment en un RFC o no. I, si resulta suficientment interessant, aquest podria convertir-se en un estàndard d'Internet. La redacció d'aquests documents es fa en anglès seguint una estructura específica i un format de text ASCII.

Cada RFC té un títol i un nombre assignat, que no pot repetir-se ni eliminar-se encara que el document es quedi obsolet. Així doncs, cada protocol dels que existeixen avui en dia a Internet té assignat un RFC que el defineix, i possiblement altres RFC addicionals que l'amplien.

Existeixen diverses categories, de forma que els RFC poden ser de tipus informatiu, propostes d'estàndards nous o històrics per a fer revisions d'algun RFC obsolet.

Cal tenir present que abans de que un document tingui la consideració d'RFC, ha de seguir un procés molt estricte per assegurar la seva qualitat i coherència. Quan aconsegueix això, pràcticament ja se'l pot considerar un protocol formal al que probablement se li posaran poques objeccions, pel que el sentit del nom de *Request For Comments* (petició de comentaris) ha quedat pràcticament obsolet, donat a que les crítiques i suggeriments es donen a les fases anteriors. Tot i això, per raons històriques es manté el nom de RFC.

2.3. Terminologia

En aquest sub-apartat es donarà una definició dels conceptes més usats Fent una definició dels conceptes més usats

- **Bitmap:** És tracta d'un mapa de bits per reflectir la disponibilitat de les unitats de transmissió més petites en *chunks*. Es pot descriure també generalment com la disponibilitat de contingut d'un *peer*. Els *peers* d'un eixam necessiten intercanviar l'informació de mapa de bits per conèixer d'on obtenir les unitats de dades que els hi interessen.

- **Buffer map:** Un mapa per indicar quins *chunks* té un *peer* actualment al seu *buffer* i pot compartir amb altres *peers*. El mapa de *buffer* pot incloure l'offset (l'identificador del primer *chunk* emmagatzemat pel *peer*), la longitud del mapa de *buffer* i una cadena de zeros i uns indicant quins *chunks* estan disponibles. Reflecteix la disponibilitat de contingut d'un *peer*.

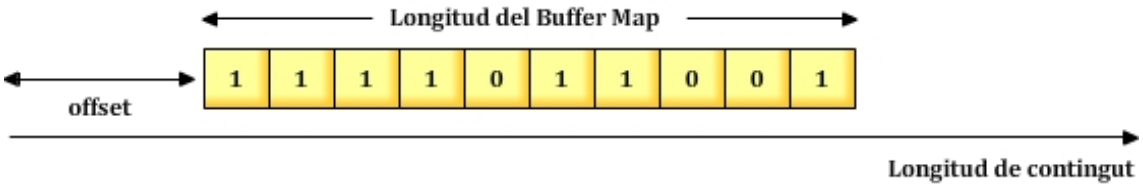

Fig. 2.1.- Esquema representatiu del Buffer Map

- **Chunk:** És l'unitat bàsica d'un *streaming* particionat, que s'usa per un *peer* amb el propòsit d'emmagatzemar, anunciament i intercanvi entre *peers*.

- **Chunk ID:** Un identificador de *chunk* per a uns certs recursos que mostra la posició, o l'*slot* de temps, del *chunk* a tot el fitxer o del *live streaming*. Un *peer* informa al *Tracker* de quins *chunks* estan mantinguts activament al seu *buffer* enviant els *Chunk IDs* d'un fitxer per un cert eixam.

- **Content Distribution Network (CDN) Node:** Un node CDN fa referència a una entitat de la xarxa que usualment es desplega a l'extrem de la xarxa per emmagatzemar el contingut proporcionat pels servidors originals, i servir contingut als clients localitzats a prop seu topològicament.

- **Live *streaming*:** Es tracta de l'escenari on tots els clients reben contingut *streaming* pel mateix esdeveniment en curs. Els retards entre els punts de play dels clients i els de la font d'*streaming* són petits.

- **P2P cache:** Una *caché* P2P fa referència a una entitat de la xarxa que captura tràfic P2P a la xarxa, i també com transparent o explícitament un *peer* distribueix contingut a altres *peers*.

- **P2P *streaming* protocols:** Els protocols d'*streaming* P2P es refereixen a múltiples protocols com *streaming protocol*, *resource discovery*, *streaming data transport*, etc. que són necessaris per construir un sistema d'*streaming* P2P.

- ***Peer*/PPSP *Peer*:** Un *peer* és un participant a un sistema *streaming* P2P. El participant no només rep contingut *streaming*, sinó que també emmagatzema contingut per compartir amb altres participants.

- **Peer list:** Una llista amb identificadors de *peer* (*Peer ID*) que estan al mateix eixam mantingut pel PPSP *Tracker*. Un *peer* ha de demanar la llista de *peers* d'un eixam al *Tracker* per saber quins *peers* tenen el contingut desitjat.

- **PPSP:** PPSP fa referència als protocols clau de senyalització entre varis components del sistema d'*streaming* P2P, incloent el *Tracker* i els *peers*. PPSP forma part dels protocols d'*streaming peer to peer*.

- **Swarm:** *Swarm*, o eixam en català, es refereix a un grup de clients que comparteixen el mateix contingut a un moment determinat.

- **Swarm ID:** Identificador per a un cert eixam. Es fa servir per descriure un recurs específic compartit entre *peers*.

- ***Tracker*/PPSP *Tracker*:** Un *Tracker* es refereix a un servidor de directori que manté llistes de *peers* amb la disponibilitat de *chunks* que tenen aquests per un canal o fitxer d'*streaming* específic, i respon peticions dels *peers* per enviar llistes de *peers*.

- **Usage Type:** Informació usada per identificar el tipus del contingut compartit. Actualment hi ha dos tipus d'*Usage Type* a PPSP, *Live streaming* i VoD. PPSP serà ampliat en un futur per suportar altres tipus.

- **Video on Demand (VoD):** Escenari on diferents clients poden veure diferents parts del contingut mèdia enregistrat i emmagatzemat durant events passats.

2.4. Disseny de l'arquitectura del protocol

2.4.1. Problemàtica

Abans d'entrar al que seria el disseny de l'arquitectura del protocol PPSP, cal donar un cop d'ull als diferents problemes[6] que comporten els sistemes de senyalització propietaris per les aplicacions d'*streaming* P2P.

2.4.1.1. Dificultats dels ISPs a la implementació de cachés de P2P

Davant de múltiples aplicacions d'*streaming* P2P, els ISP estan assistint a una tensió de tràfic molt gran a la seva xarxa troncal i als ports d'interconnexió de xarxes. La *caché*

de P2P s'usa per reduir aquest tràfic, emmagatzemant de forma dinàmica el contingut d'*streaming* al que s'accedeix freqüentment (en granularitat de *chunk* o de fitxer, segons convingui). Això es fa servir molts cops als extrems o als punts d'interconnexió entre xarxes. No obstant, els nodes de *caché* necessiten executar DPI per poder identificar els diferents sistemes d'*streaming* P2P. El fet de tenir múltiples protocols d'*streaming* P2P propietaris que vagin canviant, crea la necessitat de que la *caché* de P2P actualitzi la seva llibreria de coincidències constantment, de forma que s'incrementa el cost de l'operador de forma dràstica.

Amb PPSP, la *caché* de P2P pot detectar aplicacions d'*streaming* P2P de forma molt més senzilla. Això fa que la càrrega de treball del ISP es vegi reduïda de forma dràstica.

2.4.1.2. Dificultats a la construcció d'infraestructures d'entrega d'*streaming* obert

El futur d'Internet està centrat al contingut ja que l'immensa majoria de l'ús actual d'Internet, fins a més d'un 90% del tràfic, consisteix en dades. La majoria dels treballs centrats en el contingut estan buscant construir una infraestructura de distribució de contingut global i oberta.

A les xarxes centrades en contingut no hi ha cap dubte de que les dades referents a l'*streaming* P2P seran una porció molt important del total de la càrrega.

Si els múltiples protocols propietaris actuals continuen funcionant, existiran molts sistemes específics i independents per entregar un ampli contingut d'*streaming*.

Això porta més càrregues per identificar i compartir el mateix contingut, l'increment de l'emmagatzemament, el cost de l'enviament i el manteniment dels nodes entremitjos per contingut repetit i pel mateix contingut, moltes dades a de forma generalitzada a través d'Internet. Tot això incrementarà de forma definitiva el cost de la distribució d'*streaming* i causarà una possible congestió a la xarxa.

A una infraestructura d'entrega d'*streaming* P2P obert, s'ha d'involucrar als diferents participants a la cadena de distribució que va des de la font (o proveïdors actuals d'*streaming* P2P), infraestructura (per exemple CDN) i la banda de descarrega (*cachés* o *peers* terminals).

Amb el protocol PPSP, els nodes de CDN poden ser designats per inter-operar amb els protocols estàndards només, reduint d'aquesta forma la negociació cas per cas entre els proveïdors font i els diversos proveïdors de CDN.

2.4.1.3. Dificultats en un entorn mòbil i wireless

Els entorns mòbils (mobilitat) i *wireless* s'estan tornant elements cada cop més importants per donar suport a les implementacions futures d'Internet. Actualment hi ha més i més usuaris mòbils o que fan servir *wireless*. De fet, es preveu que cap a finals de 2012, el nombre d'usuaris mòbils d'Internet, sobrepassi el nombre d'usuaris fix a Xina.

Juntament amb aquesta tendència, l'*streaming* mòbil s'està convertint en un oferiment clau. A Korea, el nombre de subscriptors de TV mòbil ha arribat als disset milions, el que representa un terç del total dels usuaris mòbils. Durant els jocs olímpics de Beijing a l'any 2008, més d'un milió d'usuaris usava aquest servei de TV mòbil.

Tot i que els dispositius mòbils poden ser *peers* amb millor ample de banda i una major freqüència de CPU, més capacitat d'emmagatzemament així com memòria, que varis anys enrere, els *peers* mòbils i *wireless* poden tenir en general majors dificultats per suportar *streaming* P2P amb connexions de xarxa relativament més baixes (sobre tot a l'enllaç de pujada) i inestables, i menor energia constant el que els ordinadors de sobretaula. I el que és pitjor, els protocols actuals estan dissenyats principalment per l'Internet fix i no poden adreçar aquestes dificultats.

2.4.1.4. Terminal physical resource starvation

Protocols privats poden requerir un terminal per instal·lar divers i diferent software al mateix temps. Per exemple, un usuari instal·la *CBox* per programes CCTV[7] i *PPLive*[8] per pel·lícules Japoneses o Coreanes. En qualsevol cas, pot ser difícil instal·lar múltiples clients en un únic *peer* amb recursos restringits, com pot ser un node mòbil. Les limitacions de CPU, emmagatzemament i memòria, molts cops limiten el nombre total d'aplicacions instal·lades així com els processos i *threads* que poden córrer de forma concurrent.

Els protocols estàndards redueixen la complexitat de que coexisteixin múltiples sistemes tancats i creen la possibilitat d'usar un únic client de *software* acomodant diferents sistemes.

2.4.2. Arquitectura

Hi ha múltiples solucions proposades d'*streaming* P2P. Algunes s'usen a la pràctica mentre que d'altres són populars en teoria. La idea bàsica de l'*streaming* P2P és dividir l'*stream* en diferents *chunks* que els *peers* s'encarreguen de transmetre per la xarxa entre ells. Les solucions es poden categoritzar en sistemes basats en arbre o basats en

malla, segons la seva distribució. A la primera de les solucions, la construcció i el manteniment de l'arbre pot fer-se de forma centralitzada o distribuïda. En l'*streaming* P2P basat en malla, no es necessària un manteniment de topologia. La feina és com localitzar i recuperar dades en temps real des de múltiples fonts amb diferents *chunks* per *streaming*. Per realitzar això, cada interlocutor ha de localitzar activament *caché* i intercanviar *chunks* amb altres *peers* sota l'orientació del *Tracker*.

El major desafiament a l'*streaming* P2P basat en arbre és doble: Primer, l'*streaming* basat en arbre no es pot recuperar suficientment ràpid per gestionar la rotació freqüent dels *peers*. Segon, els *peers* dels nivells superiors de l'arbre necessiten tenir una capacitat d'*upload* suficient per suportar l'*streaming* dels seus fills. De no ser així, el rendiment es deterioraria de forma espectacular. Pel contrari, l'*streaming* P2P basat en malla s'usa molt per superar aquests inconvenients a l'hora de fer una implementació real, com és el cas de *PPLive* o *PPStream*[9].

Als sistemes *streaming* P2P, hi ha varis eixams on cada un d'aquests eixams conté un grup de clients compartint el mateix contingut *streaming* (un canal, un fitxer *streaming*, etc) a un cert instant. Aquests clients és el que s'anomena *peers*, de forma que cada clients no només rep contingut *streaming*, sinó que també emmagatzema i puja aquest contingut *streaming* a altres clients. En un sentit ampli de la infraestructura global d'entrega de continguts, els *peers* poden incloure múltiples tipus d'entitats com aplicacions d'usuari final, *cachés*, nodes de CDN i/o altres dispositius finals. Per tant, les funcions bàsiques d'un sistema *streaming* P2P inclouen:

1. Mantenir la informació sobre quins *peers* hi ha a cada eixam a algun directori de serveix. El que es coneix com a *Tracker*.

2. A cada eixam, intercanviar informació sobre la disponibilitat de contingut (quins *chunks* té cada *peer*) entre *peers* o entre el *Tracker* i els *peers*.

3. A cada eixam, intercanviar el contingut actual entre *peers*.

Tal i com es pot veure a la *Fig. 4.2*, els fluxos d'informació generals a un sistema *streaming* P2P inclouen quan un *peer* vol rebre contingut *streaming* i quan un *peer* vol compartir contingut *streaming* amb altres *peers*.

Al primer dels casos, primer el *peer* obté una llista de *peers* de l'eixam corresponent a través del *Tracker*. Un eixam pot ser indexat per un *Channel ID*, *streaming file ID*, etc. Tot seguit, el *peer* intercanvia la seva disponibilitat de contingut amb els *peers* de la llista de *peers* obtinguda. Finalment, el *peer* identifica els *peers* amb contingut desitjat i

fa les peticions corresponents pel contingut dels *peers* identificats. Al segon cas, quan un *peer* vol compartir contingut *streaming* amb altres *peers*, el *peer* envia informació cap al *Tracker* sobre els eixams als que pertany, així com el seu estat d'*streaming* i la disponibilitat de contingut.

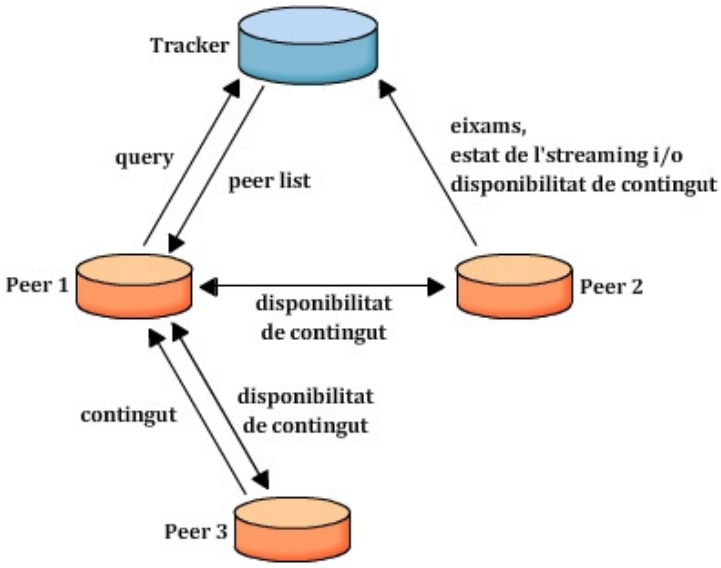

Fig. 2.2.- Fluxos d'informació a un sistema streaming P2P

2.4.3. Integració amb protocols existents

PPSP no serà un protocol independent, és a dir, que actuarà conjuntament amb altres protocols per realitzar determinades accions ja definides. Un dels avantatges d'un protocol estàndard és que es pot considerar explícitament la seva interacció amb altres protocols i dissenyar-ho per un sistema integrat. En particular, PPSP interactuarà amb els protocols RELOAD, ALTO i RTSP, on P2PSIP, ALTO i MMUSIC són els grups de treball més relacionats.

2.4.3.1. Integració amb RELOAD

El protocol RELOAD, definit per P2PSIP, tracta amb la localització de recursos a una comunicació extrem a extrem. El procés iteratiu i recursiu d'enrutament a RELOAD és lleugerament diferent de PPSP. És a dir, les dades emmagatzemades a RELOAD és informació de perfil d'usuari i el que fa la petició sap exactament que són aquestes dades. Mentre que en PPSP, hi ha molts *peers* emmagatzemant diferents peces de dades. La única cosa que un *peer* necessita saber són les metadades del vídeo. Es necessita un protocol per comunicació amb altres *peers* per obtenir les dades reals de forma ràpida.

La major diferència entre P2PSIP i PPSP radica als seus requeriments d'eficiència de recerca diferents. PPSP requereix de la recuperació de dades en temps real amb un temps estricte. La topologia basada en DHT suportada a RELOAD pot no ser valida per l'emmagatzemament de la informació de *peer* i la recuperació d'aquesta. Hi ha algunes raons tal i com es veu tot seguit:

Primer de tot, la quantitat de *chunks* registrats al mateix temps és gran, de l'ordre de varis centenars a la *caché*. En segon lloc, l'informació de *chunk* emmagatzemada serà invàlida molt ràpid, varis minuts a un *streaming* en viu. A diferència de l'*streaming* VoD, l'informació de *chunk* no serveix de res si el temps requerit és prou gran com per excedir el llindar de temps del contingut *time-out* de *caché*. Si cada *peer* publica contínuament la seva informació de *chunk* dinàmicament segons canvia, una gran càrrega es generarà.

Tot i que P2PSIP no encaixa per l'organització de *peers* en *streaming*, pot ser desplegat en un entorn PPSP fins a cert punt.

2.4.3.2. Integració amb ALTO

ALTO té com objectiu definir un protocol que es doni compte del tot el tràfic local per aplicacions P2P. La finalitat d'una solució ALTO seria per ajudar als *peers* a trobar els millors orígens i els millors destins pels fluxos mèdia que reben i envien. En aquest sentit, l'*streaming* P2P en viu es pot beneficiar de *ALTO service*[10].

Com s'ha comentat més d'una vegada, el tràfic *streaming* P2P és molt gran a la xarxa troncal de Internet. PPSP ha de considerar formes senzilles de reduir el tràfic que creui la xarxa troncal amb la intenció de controlar el cost de transmissió. ALTO és un bon candidat per fer-ho.

ALTO dóna l'oportunitat de que un tercer proporcioni informació de localització dels *peers* basant-se en la premissa de que aquest tercer té el coneixement de la topologia de la xarxa. A més, poden haver-hi significats de localitzacions de tràfic extra, fent per exemple que els *peers* puguin agrupar als *peers* propers de forma senzilla i sense consum important d'ample de banda. Es pot usar aquest mecanisme a la selecció de *peers* per reduir el tràfic.

2.4.3.3. Encapsulant RTP

A primera vista, la funció de PPSP és similar al protocol tradicional de control *streaming* basat en client/servidor, RTSP[11]. Però el fet és que RTSP no inclou els problemes que PPSP té.

A RTSP, la idea es controlar l'*streaming* (per dir-ho d'alguna forma, el que seria fer PLAY, PAUSE, etc.). En canvi, PPSP es centra a la senyalització dels *peers* pel descobriment de recursos en temps real, ajuntar i sincronitzar, en comptes de com realitzar el control de l'*streaming*.

Per altra banda, tenint en compte la prevalença de RTSP, també es pot reutilitzar en clients sense necessitat de tenir instal·lat software compatible amb PPSP per usar un servei d'*streaming* P2P.

La idea bàsica d'això és configurar un node amb *Proxy*, que pugui actuar com a servidor RTSP que és en realitat un *peer* de PPSP.

2.5. Abast del protocol PPSP

El rol bàsic de PPSP és descobrir contingut en temps real distribuït a un escenari *streaming peer to peer*, és a dir, que la clau de PPSP és trobar quin *peer* té cada contingut. La part fonamental de PPSP és un conjunt de protocols de senyalització per implementar la negociació entre *peers* sobre el contingut d'informació de cada *peer* i qualsevol altra informació relacionada amb proporcionar el servei.

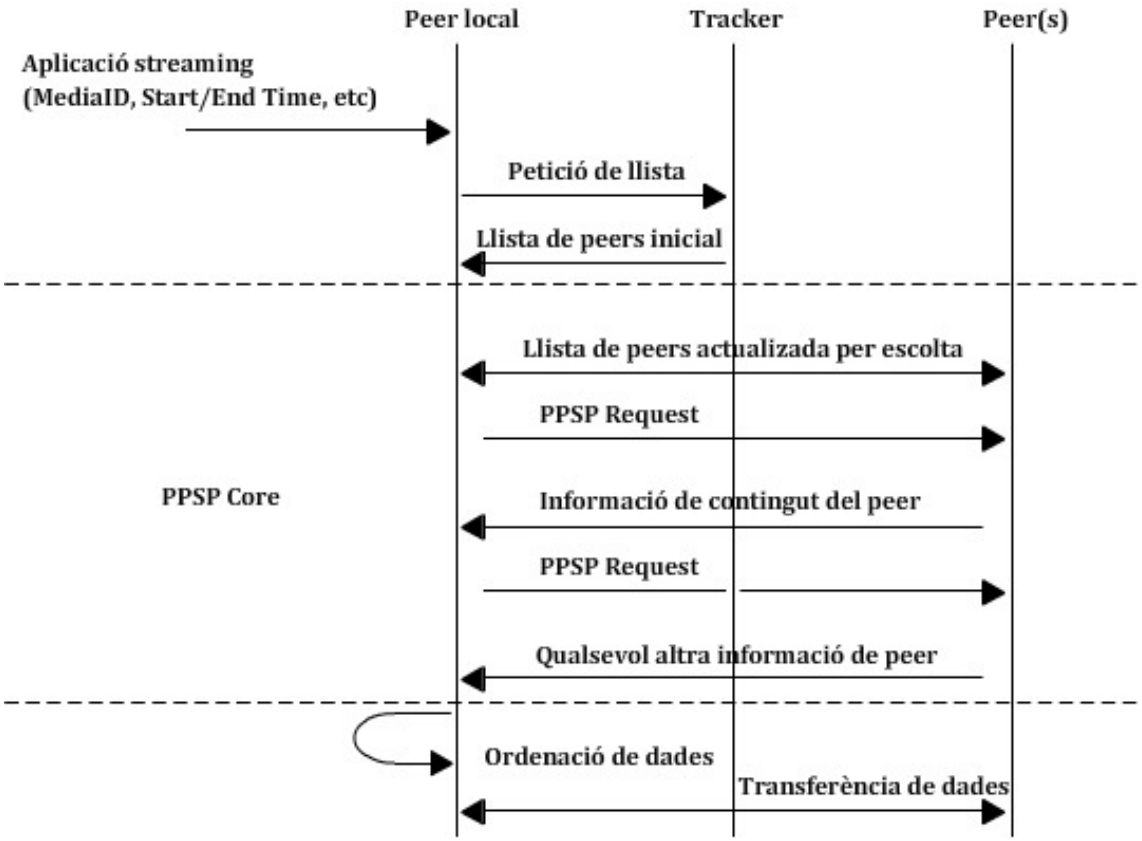

Fig. 2.3.- Esquema de comunicació a la part principal de PPSP

A dins de l'abast que ha d'abastar el protocol, es consideren els components següents:

- **Tracker Communication:** Comunicació amb el *Tracker*, de forma que cada *peer* pugui obtenir la llista de *peers* del *Tracker* i/o proporcionar la disponibilitat de contingut al *Tracker*.

- **Peer Communication:** Comunicació entre *peers*, de forma que cada *peer* pugui intercanviar disponibilitat de contingut amb altres *peers* i fer peticions de contingut d'aquests.

- **Report:** Aquest és un component que permet als *peers* reportar el seu estat d'*streaming* cap al *Tracker*. Aquesta informació inclourà *swarm IDs* per mostrar els eixams als que el *peer* té participació, llista de *chunks* per cada eixam per veure l'actual disponibilitat de contingut al *peer*, la capacitat de tràfic d'entrada i de sortida, quantitat de *peers* veïns, el grau de salut dels *peers* i altres paràmetres d'*streaming*.

A més, d'aquests components, PPSP inclou el *PPSP Tracker Protocol*, un protocol de senyalització entre *Trackers* i *peers* a PPSP, i el *PPSP Peer Protocol*, un protocol de senyalització entre *peers* a PPSP.

2.5.1. Sistemes d'streaming P2P: Pull-based i Push-based

Bàsicament, hi ha dos tipus de sistemes d'*streaming* P2P: els sistemes *pull-based* i els *push-based*. Als sistemes d'*streaming* P2P *pull-based*, un *Tracker* centralitzat o diversos *Trackers* distribuïts mantenen informació sobre quins *peers* estan a quins eixams i responen les peticions dels *peers* sobre aquesta informació amb una llista de *peers*. Un cop rebut el missatge, el *peer* pot connectar-se amb els diferents candidats dins d'un eixam, intercanviar la seva disponibilitat de contingut a la seva memòria (en funció de si es *streaming* en temps real o *Video on Demand*) amb altres *peers* i llavors obtenir les dades d'*streaming* desitjades. L'eixam té una topologia de malla. Els avantatges del mode *pull-based* són la seva robustesa davant el *peer churn* i la latència acceptable per una reproducció suau i sense problemes.

D'altra banda, a un sistema d'*streaming* P2P *push-based*, hi ha un node principal que manté la topologia, com pot ser una topologia en forma d'arbre. Els *peers* en aquesta topologia comparteixen el mateix interès en el contingut. Tant la senyalització com la distribució de dades estan ambdues basades en aquesta mateixa topologia. Per un programa o fitxer de vídeo, el *peer* fa una petició cap al node principal per conèixer la seva localització per unir-se i el node principal respon amb una llista de *peers*,

segurament amb l'ordre de connexió recomanat. Després de rebre la llista de *peers*, el *peer* es pot connectar amb els *peers* candidats per poder ser un node a cert lloc de la topologia i rebre les dades a través d'aquesta topologia sense la necessitat de intercanviar disponibilitat de contingut amb els seus companys. En aquest sentit, el node principal actua com si fos el *Tracker*. El *push-based mode* té com a avantatges una menor latència però, per contra, la topologia és fràgil al *peer churn*. Diversos sistemes pràctics usen aquest mode. Un mode més pràctic és fer un híbrid dels modes *pull-based* i *push-based*, anomenat *pull-push-based mode*, on els *peers* intercanvien disponibilitat de contingut amb els seus companys per recuperar dades.

Com ja s'ha comentat, hi ha principalment dues entitats importants a dins de l'*streaming* P2P: *Trackers* i *peers*. PPSP està destinat a estandarditzar els protocols de senyalització en arquitectures basades en *Tracker* per poder suportar tant *streaming* en viu com VoD.

S'ha de tenir present que al mode *pull-based* i al mode híbrid *pull-push-based*, tant el *Tracker Protocol* com el *Peer Protocol* poden ser utilitzats, mentre que en el p*ush-based mode*, només s'usarà el *Tracker Protocol*. Aquests protocols es veuen breument a continuació i més profundament als apartats 4 i 5 d'aquesta memòria.

2.5.2. Protocols de PPSP

S'ha proposat estandarditzar protocols a PPSP que permetin els components de comunicació del *Tracker* (*Tracker Comunication*) i del *peer* (*Peer Comunication*) a la capa de comunicació, així com el component de *Report* a la capa d'informació. Aquests protocols, anomenats en conjunt PPSP, són mecanismes que exerceixen dos papers importants: els de *Tracker* i *peer* a un sistema *streaming* P2P.

Aquests protocols de senyalització, en essència, tenen l'objectiu d'estandarditzar els mecanismes d'intercanvi de l'informació de contingut entre diferents dispositius a un sistema *streaming* P2P. Cal tenir present que PPSP és només una part dels protocols *streaming* per P2P, que es desenvolupen paral·lelament al treball fet a PPSP.

Degut a que els components de bootstrap, registre i estadístics són mecanismes considerats *out-of-band* pels processos *streaming*, no s'inclouen a dins de l'abast de PPSP. Actualment, tant les capes de transport, *play-out* i aplicació a un sistema *streaming* P2P estan també fora de l'àmbit de PPSP.

No obstant, PPSP inclou el PPSP *Tracker Protocol* i el *PPSP Peer Protocol*.

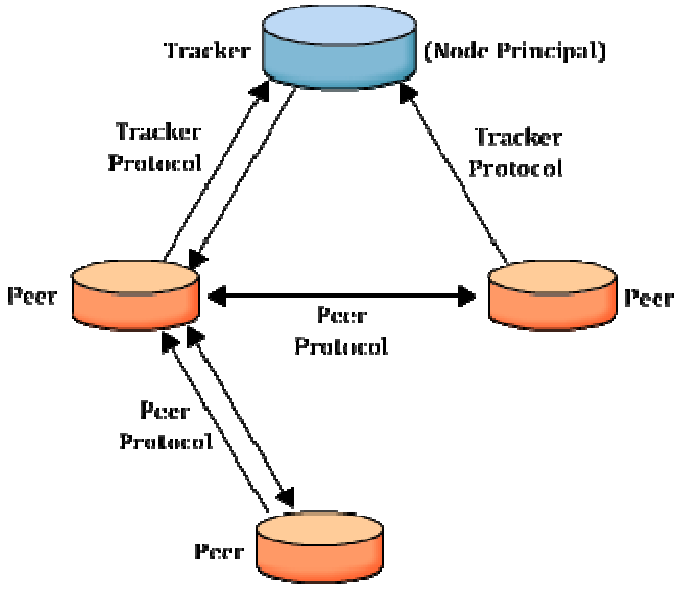

Fig. 2.4. - Arquitectura de sistema de PPSP

2.5.2.1. PPSP *Tracker* Protocol

PPSP Tracker Protocol[12] és un protocol de senyalització entre els *Tracker* i *peers* de PPSP. El protocol fa una definició del format i codificació estàndard de la informació entre *peers* i *Tracker* a PPSP, així com proporcionar la llista de *peers*, la disponibilitat de contingut, l'estat de l'*streaming* incloent el temps *online*, l'estat dels enllaços, capacitat dels nodes i altres paràmetres d'*streaming*.

Per aquesta missió, té definits una sèrie de missatges que determinen entre els *peers* i el *Tracker* com els *peers* de PPSP reporten l'estat de l'*streaming* i com es fan les peticions al *Tracker*, així com les respostes que es donen des del *Tracker* a aquestes peticions.

2.5.2.2. PPSP *Peer* Protocol

PPSP Peer Protocol[13] és un protocol de senyalització entre els *peers* a PPSP. No hi ha interactuació amb el Tracker mitjançant aquest protocol. Aquest protocol fa una definició del format estàndard i la codificació de la informació entre els *peers* a PPSP, així com la descripció de *chunks*. A més, haurà de fer una definició dels missatges estàndards que intercanviaran els *peers* entre ells, que definiran com els *peers* informen de la disponibilitat de *chunks* als altres, així com la senyalització per fer peticions de *chunks* a altres *peers*.

2.5.3. Service Types

PPSP serveix com a tecnologia habilitadora i com eina per la construcció de diversos sistemes d'*streaming* P2P. Els tipus de servei suportats pels sistemes d'*streaming* P2P actuals inclouen *live streaming* i *video on demand*.

A *live streaming*, o *streaming* en viu, tots els *peers* estan interessats al contingut mèdia que procedeix d'un esdeveniment que està en marxa, el que vol dir que tots els *peers* comparteixen aproximadament el mateix contingut *streaming* a un punt determinat del temps. A *live streaming*, alguns *peers* emmagatzemaran el contingut *live media* per una posterior distribució, que es coneix com a TSTV (*time-shift TV*) on el contingut mèdia emmagatzemat està separat en *chunks* i distribuït com si es tractés d'un sistema *Video on Demand*.

A VoD, en canvi, diferents *peers* poden veure diferents parts del contingut mèdia enregistrat i emmagatzemat durant un esdeveniment passat. En aquest cas, cada *peer* continua preguntant a altres *peers* quins *chunks* d'aquest mèdia estan emmagatzemats en quins *peers*, i agafa el contingut requerit d'algun dels diferents *peers*.

2.5.4. Fora de l'abast del protocol

Hi ha diversos temes que estan fora de l'abast del grup de treball del protocol PPSP, és a dir, que no s'han de tenir presents a l'hora del desenvolupament del protocol. Entre aquests temes es troben:

- Mecanismes de transport per la transmissió de dades entre *peers* a PPSP, que es faria usant protocols de transport com TCP o UDP, *unicast* o *multicast*, etc.
- Mecanismes de codificació de xarxa a l'*streaming* P2P. La codificació de xarxa es pot usar a l'*streaming* P2P, però el seu suport està més enllà de l'àmbit del *charter* actual.
- Mecanismes de *Play-out* incloent el control de l'*streaming* (*play*, *pause*, tirar endavant, tirar enrere, etc.).
- Algorismes que poden ser diferents en diferents implementacions, com per exemple la selecció de *chunks* o les estratègies locals d'emmagatzemament.
- Qüestions de recerca de tipus relacionades amb *streaming* P2P. El *working group* sotmetrà dites qüestions al *P2P Research Group* del IRTF o altre grup apropiat.

2.6. Buffer Map

Basant-se al funcionament d'altres sistemes *streaming* P2P, es pot fer una aproximació del *buffer map* de PPSP. Un *peer* pot demanar, sobre una connexió TCP, el *buffer map* de qualsevol *peer* de la seva llista de *peers* actuals. Després de que el *peer* rebi el *buffer map* del *peer* al qui li ha demanat, el primer pot demanar un o més *chunks* dels que aquest últim *peer* ha anunciat al seu *buffer map*. Un *peer* pot descarregar *chunks* de múltiples *peers* diferents de forma simultània. La llista de *peers* es va actualitzant periòdicament de forma que els *peers* dels que poder descarregar els *chunks* necessaris seran diferents amb el temps.

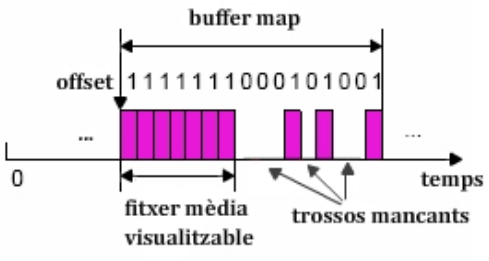

Fig. 2.5. - Buffer Map de PPSP

2.7. Casos d'ús de PPSP

2.7.1. Disposició Worldwide de serveis live streaming P2P obert

Els proveïdors poden expandir de forma fàcil l'escala de *broadcasting* amb PPSP. A la figura següent, les interaccions entre el *Tracker* del proveïdor A i els super-nodes (SN) dels proveïdors B i C, poden ser normalitzades mitjançant l'ús del *Tracker Protocol*; i el *Peer Protocol* pot ser usat entre la difusió de super-nodes i/o *peers* a diferents proveïdors.

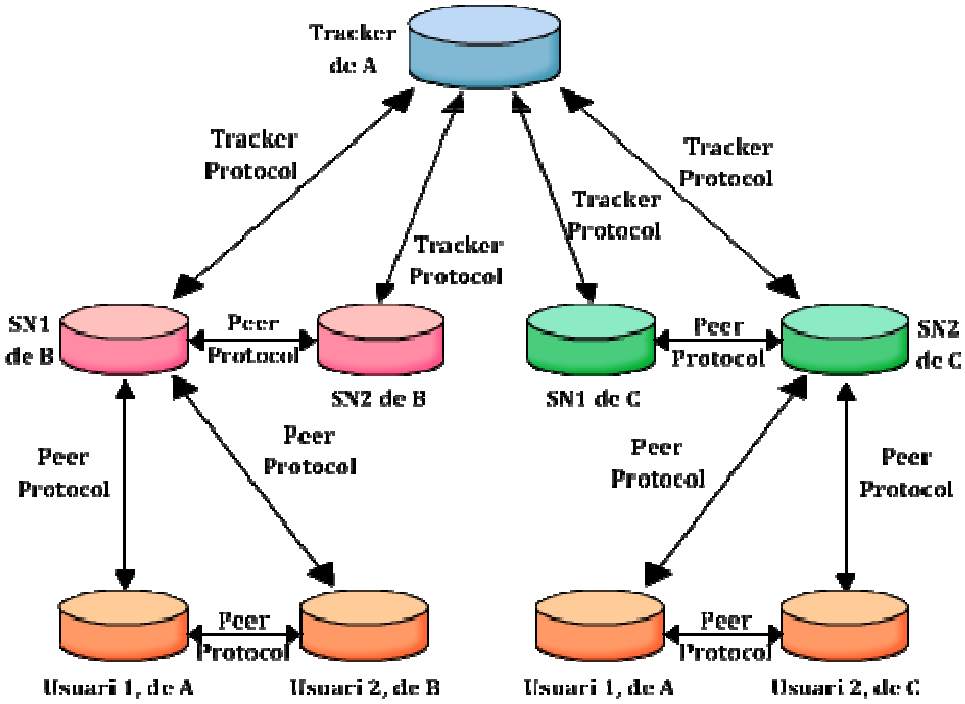

Fig. 2.6.- Interaccions entre proveïdors cooperatius

2.7.2. CDN amb suport per streaming P2P

Aquest escenari és similar al cas anterior, però involucra més tipus de participants: a més de proveïdors d'*streaming* P2P, també estan involucrats a l'entrega proveïdors que no són d'*streaming* P2P i proveïdors de CDN.

Els substituts CDN poden actuar com els super-nodes de diferents proveïdors d'*streaming* P2P amb PPSP. Les interaccions entre aquestes entitats de xarxa són les mateixes que a la figura 2.6. Els proveïdors d'*streaming* P2P poden llogar recursos CDN per proporcionar millors serveis que assegurin la QoS per usuaris VIP, diferenciant-los dels serveis proveïts per *peers* ordinaris.

Un altre cas és en el que els proveïdors CDN ofereixen distribució d'*streaming* P2P amb PPSP. Això succeeix freqüentment per un operador o un proveïdor de CDN al construir un nou sistema CDN que suporti aplicacions d'*streaming* amb cost reduït. Aquest servei és molt útil per a petits proveïdors d'*streaming* que no tenen massa recursos (diners, etc.) per distribuir el seu *stream* arreu del mon. També pot ser usat per un operador per llençar un servei d'*streaming* auto-operat des del principi, és a dir, des de zero.

2.7.3. PPSP amb suport d'streaming cross-screen en entorns heterogenis

A l'escenari de la figura següent, un PC un *Setbox*/TV o un terminal mòbil, pertanyents a xarxes fixes o mòbils, treballen conjuntament compartint el contingut que emmagatzemen i finalitzant l'entrega de l'*streaming*.

Utilitzant el protocol PPSP, els *peers* poden identificar els diferents tipus de xarxes, les habilitats dels *peers* i arribar a conèixer quin contingut tenen altres *peers*, inclús en condicions de xarxa, com la comentada anteriorment i tal com es pot veure a la figura següent. Això jugarà un paper important a la compartició entre *peers* heterogenis.

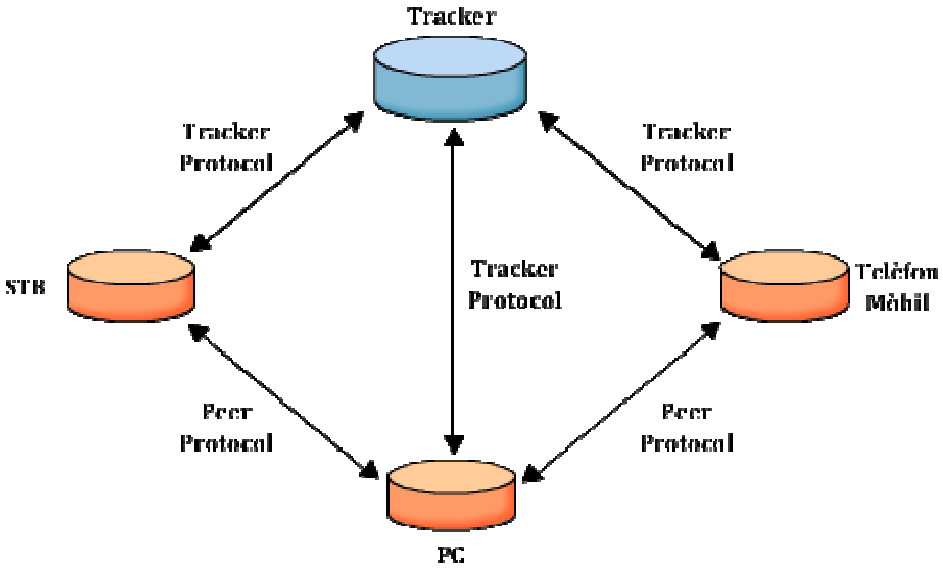

Fig. 2.7.- Streaming P2P heterogeni interactuant amb PPSP

2.7.4. Suport d'streaming P2P a xarxes mòbils

En un entorn mòbil com pot ser 3G o 4G, amb l'increment d'ample de banda i les capacitats dels terminals mòbils intel·ligents, l'*streaming* P2P és més senzill de portar a terme que abans. Cal adonar-se de que els terminals mòbils no tenen obligació de convertir-se en *peers*. Els *peers* de xarxa que estan desplegats pels ISPs o operadors i els *peers* mòbils amb connexions WiFi, són seleccionats de manera més freqüent. Per exemple, a 3GPP, hi ha un ítem de treball P2P CDS als requeriments dels operadors mòbils per usar preferentment equipaments d'extrem de xarxa per actuar com *super-peers* quan no hi ha suficients *peers* per escollir per realitzar l'*streaming* P2P. Degut a que estan desplegats pels operadors, la seva estabilitat i capacitat d'emmagatzemament estan millor garantides que pels *peers* ordinaris.

En aquest cas, el *PPSP Tracker Protocol* identificarà els tipus de xarxa i les dinàmiques, així com els tipus de terminal i retornarà *super-peers* a la llista de *peers* cap aquests *super-peers* i als *peers* mòbils normals. Si els terminals mòbils no poden ser escollits com a *peers*, simplement rebran data dels *super-peers* sense contribuir amb cap dada cap a altres.

2.7.5. Servei caché amb suport d'streaming P2P

Tal com s'ha comentat anteriorment, el desplegament de nodes de *caché* als extrems de la xarxa pot decrementar de forma molt gran el tràfic entre xarxes i incrementar l'experiència dels usuaris als serveis d'*streaming*.

Amb PPSP, els nodes de *caché* poden identificar el tipus d'*streaming* P2P inclús si inclou diferents aplicacions, memòria *caché* dels continguts freqüentment visitats i reporta el que la seva *caché* té cap al *Tracker* del proveïdor com un *peer* normal i serveix a altres *peers* que facin peticions en la compartició de dades tal i com es veu a la figura 2.8. Els nodes de *caché* no necessiten actualitzar les seves llibreries quan noves aplicacions s'introdueixen, el que permet als nodes de *caché* tenir menys cost per suportar més aplicacions.

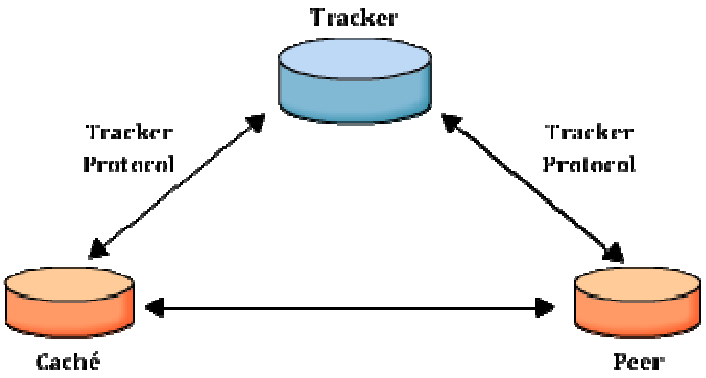

Fig. 2.8.- Servei caché amb suport d'streaming amb PPSP

3. Requeriments de PPSP

L'objectiu del treball de PPSP és estandarditzar els protocols de senyalització clau que fan referència tant a *Tracker* (*PPSP Tracker Protocol*) com a *peer* (*PPSP Peer Protocol*) a un sistema *streaming peer to peer*. Aquest conjunt de protocols és el que es coneix com *Peer to Peer Streaming Protocol*.

La computació del *peer* to *peer* ha sigut usada exitosament a molts camps, des de comunicació *one to one* com pot ser Veu sobre IP (VoIP) o missatgeria instantània (IM), a comunicació *one to many* com pot ser l'*streaming*, compartició d'arxius o el joc online. A l'àrea d'*streaming*, la popularitat de les tecnologies *streaming* P2P en temps real i de *video on demand* (VoD) ha sigut demostrada per *PPLive*, *PPStream*, *UUSee*, *Pando*, entre d'altres. Si agafem *PPLive* per exemple, aquest té uns 5 milions d'usuaris online al mateix temps per *streaming* en temps real. Les aplicacions d'*streaming* P2P cada cop ocupen més i més tràfic a Internet. D'acord amb les estadístiques del major proveïdor de servei d'Internet (ISP) de China, el tràfic generat per les aplicacions d'*streaming* P2P sobrepassa el 50% del tràfic total de *backbone* durant l'hora punta a l'any 2008.

Donat l'increment de la integració de l'*streaming* P2P a la infraestructura global de distribució de contingut, la necessitat d'un protocol d'*streaming* obert i estàndard s'ha fet més patent i s'ha convertit en un dels més grans components que falten al *protocol stack* d'Internet. Fins ara, múltiples protocols d'*streaming* P2P similars però propietaris han resultat de diversos esforços de desenvolupament repetitius. I més important, això ha conduit cap a unes dificultats substancials alhora d'integrar l'*streaming* P2P com un component d'una infraestructura de distribució de contingut global. Per exemple, els protocols d'*streaming* P2P propietaris no integren bé amb dispositius d'infraestructura com *cachés* i altres.

Com s'ha comentat anteriorment, l'objectiu de la creació de PPSP és el poder estandarditzar els protocols clau de senyalització que s'apliquen tant a *Tracker* com a *peer* a sistemes d'*streaming* P2P, i aquests protocols seran el que s'anomena PPSP. PPSP podrà servir com una tecnologia habilitadora, a partir d'experiències de desenvolupament de sistemes d'*streaming* P2P existents. El seu disseny permetrà que es pugui integrar els reforços de l'IETF a la localització de recursos distribuïts, localització de tràfic i mecanismes de control d'*streaming*. Permet a més l'integració efectiva amb infraestructures tals com *caché* o equipament final mòbil.

3.1. Visió general de PPSP

A l'àmbit d'aplicació de PPSP es consideren els diferents components:

- ***Tracker* Communication:** La comunicació amb el *Tracker* és un component que permet a cada *peer* obtenir la llista de *peers* des del *Tracker* i/o proporcionar disponibilitat de contingut al *Tracker*.

- ***Peer* Communication:** La comunicació entre *peers* és un component que permet a cada *peer* intercanviar disponibilitat de contingut i fer peticions de contingut a altres *peers*.

- **Report:** *Report* és un component que permet als *peers* fer peticions de l'estat de l'*streaming* al *Tracker*. L'informació pot incloure identificadors d'eixam (*swarm ID*) per mostrar els eixams als que el *peer* està involucrat activament, llista de *chunks* per cada eixam per mostrar la disponibilitat de contingut actual del *peer*, capacitat del tràfic d'entrada o de sortida, la quantitat de *peers* veïns, el grau de salut dels *peers* i altres paràmetres d'*streaming*.

Per tant, PPSP inclou el *PPSP Tracker Protocol*, un protocol de senyalització entre *Tracker* i *peers*, i el *PPSP Peer Protocol*, un protocol de senyalització entre *peers*.

El *PPSP Tracker Protocol* defineix:

- Format i codificació d'informació estàndard entre *Trackers* i *peers* a un entorn PPSP. Informació tal com: llista de *peers*, l'identificador d'eixam, informació de *chunk*, disponibilitat de contingut, estat de l'*streaming* incloent el temps online, l'estat de l'enllaç, la capacitat del node i altres paràmetres d'*streaming*.

- Missatges estàndards entre *Trackers* i *peers* definint com aquests *peers* reporten l'estat d'*streaming* i el demanen als *Trackers*, així com aquests *Trackers* responen als diferents *peers*.

D'altra banda, el *PPSP Peer Protocol* defineix:

- Format i codificació de l'informació estàndard entre els *peers*, tals com la descripció dels *chunks*.

- Missatges estàndards entre els *peers* definint com aquests *peers* anuncien la disponibilitat de *chunks* cap als altres, així com la senyalització per fer peticions de *chunks* entre *peers*.

3.2. Requeriments de PPSP

Els diferents requeriments de PPSP es poden separar en els subapartats que es veuran a continuació. Per una banda, Requeriments bàsics dels protocols PPSP (protocols de *Tracker* i de *peer*), de les entitats (*Tracker* i *peer*) i del contingut *streaming*. D'altra banda, requeriments generals pel *Tracker Protocol* i pel *Peer Protocol*.

3.2.1. Requeriments bàsics

Els protocols de *Tracker* i *peer* han de ser tant similars com sigui possible, en termes de disseny, formats de missatge i fluxos. La idea és que el *Peer Protocol* ha de ser com una extensió del *Tracker Protocol* afegint alguns tipus de missatges, o viceversa. Aquests protocols de *Tracker* i de *peer* han de permetre als *peers* rebre contingut *streaming* dins de les limitacions de temps requerides, és a dir, complir amb la característica d'*streaming*.

Cada *peer* ha de tenir un identificador únic, el *Peer ID*, a un eixam. Aquest és un requeriment bàsic per que *peer* pugui ser identificat de forma única a un eixam de forma que altres *peers* o el *Tracker* es puguin referir al identificador pel *peer*.

El contingut *streaming* ha de poder ser identificat de forma única mitjançant un identificador d'eixam (*swarm ID*). Un eixam es refereix a un grup de *peers* que comparteixen el mateix contingut *streaming*. Un *swarm ID* identifica únicament a un eixam. Aquest identificador es pot usar en dos casos diferents: el primer d'aquests és quan un *peer* demana al *Tracker* per la llista de *peers* indexada pel *swarm ID*, la segona es quan un *peer* comunica al *Tracker* els eixams als que pertany.

El contingut *streaming* també ha de permetre que es pugui particionar en *chunks*. Un *chunk* és l'unitat bàsica d'un *streaming* particionat, que s'usa per un *peer* amb el propòsit d'emmagatzemar, anunciament i intercanvi entre *peers*. Una característica clau d'un sistema *streaming* P2P és permetre fer *fetchs* de les dades de diferents *peers* de forma concurrent. Per tant, tot el contingut *streaming* ha de permetre ser particionat en *chunks* per la transmissió entre *peers*.

Cada *chunk* ha de tenir un identificador únic (*chunk ID*) a l'eixam. D'aquesta forma el *peer* pot entendre quins *chunks* estan emmagatzemats a cada *peer* i quins *chunks* han demanat altres *peers*.

A més de tot això, es recomana que els protocols de *Tracker* i de *peer* funcionin sota TCP, o com a molt sota UDP quan els requeriments no es poden satisfer amb TCP.

3.2.2. Requeriments del PPSP Tracker Protocol

El *PPSP Tracker Protocol* defineix com els *peers* reporten i demanen informació cap al *Tracker* i d'ell, i com el *Tracker* contesta a aquestes peticions. El *Tracker discovery* i la possible comunicació entre diferents *Trackers* està fora de l'abast del protocol.

El *Tracker* ha d'implementar el *PPSP Tracker Protocol* per rebre peticions i actualitzacions i reports d'estat dels *peers* periòdicament des dels *peers* i per poder enviar les respostes corresponents en cada cas.

Per la seva part, el *peer* ha d'implementar el *Tracker Protocol* per poder enviar peticions i actualitzacions i reports d'estat del *peer* de forma periòdica cap al *Tracker* i rebre les corresponents respostes.

Els missatges de petició del *Tracker* ha de permetre al *peer* que fa la petició sol·licitar la llista de *peers* del *Tracker* per un identificador d'eixam específic. Aquest missatge de petició pot incloure també els paràmetres de preferències dels *peers*, tals com el nombre de *peers* preferits de la llista de *peers* o l'ample de banda de descarrega preferit. D'aquesta forma es podrà seleccionar un grup de *peers* apropiat pel *peer* que fa la petició d'acord a les preferències.

D'altra banda, els missatges de resposta del *Tracker* han de permetre al *Tracker* oferir la llista de *peers* d'un eixam específic al *peer* que la demani.

El *Tracker* ha de poder generar la llista de *peers* amb l'ajuda de serveis d'optimització de tràfic, com pot ser el que ofereix el protocol ALTO.

El missatge d'actualització o report de l'estat del *peer* ha de tenir la capacitat d'informar al *Tracker* sobre l'activitat del *peer* a l'eixam i la informació dels *chunks* del *peer*. Aquesta informació de *chunk* ha de contenir al menys el *chunk ID*. El missatge d'actualització o report de l'estat del *peer*, a més d'això, ha de poder reflectir l'estat del *peer*.

Els canvis de l'estat del *peer* han de ser reportats cap al *Tracker* a través de missatges d'actualització o report d'estat de *peer*. Per exemple, l'estat del *peer* pot ser temps online, estat de l'enllaç físic incloent DSL/WiFi/etc, l'estat de les bateries, capacitat de processament, així com altres característiques del *peer*. Per tant, el *Tracker* és capaç de seleccionar millors *peers* candidats per fer l'*streaming*.

En alguns escenaris mòbils, l'estat del *peer* pot incloure canvi d'adreces IP.

3.2.3. Requeriments del PPSP Peer Protocol

El *PPSP Peer Protocol* defineix com els *peers* anuncien la disponibilitat de contingut d'*streaming* i intercanvien el seu estat entre ells. Aquest protocol també defineix les peticions i les respostes de *chunks* entre els *peers*. El mecanisme de transport i el control de transmissió, no obstant, estan fora de l'abast.

Els missatges de petició de disponibilitat de contingut d'*streaming* ha de permetre al *peer* sol·licitar l'informació de *chunk* d'altres *peers* de la llista de *peers*. Aquesta informació de *chunk* al menys haurà de contenir el *chunk ID*.

Els missatges de resposta a les peticions de disponibilitat de contingut d'*streaming* han de permetre al *peer* la possibilitat d'oferir la informació dels *chunks* al seu *buffer* de contingut. Aquesta informació dels *chunks* ha de contenir al menys el *chunk ID*. Aquests missatges de petició de disponibilitat de contingut també han de permetre al *peer* l'opció de sol·licitar una llista de *peers* addicional a la que ha rebut del *Tracker*, amb el mateix identificador d'eixam. És possible que un *peer* pugui necessitar *peers* addicionals per cert contingut d'*streaming*. Per tant, es permet al *peer* la comunicació amb altres *peers* a la llista de *peers* actual de forma que així es pugui obtenir una llista de *peers* addicional pel mateix eixam.

El *Peer Protocol* ha de suportar els diferents missatges d'actualització de disponibilitat de contingut *streaming* entre *peers*. Degut a un canvi dinàmic del contingut *streaming* del *buffer* a cada *peer* i la freqüència amb la que els *peers* de l'eixam s'uneixen o el deixen, la disponibilitat de contingut *streaming* entre els veïns d'un *peer* (és a dir, els *peers* coneguts per un *peer* mitjançant la llista de *peers* agafada tant de *Tracker* com de *peers*) sempre canvia i s'ha d'anar actualitzant en el temps. Aquesta actualització s'ha de fer al menys a les peticions.

D'altra banda, cada *peer* a l'eixam pot anunciar la seva disponibilitat de contingut *streaming* a altres *peers* de forma periòdica. Tot i això, els mecanismes detallats per aquesta actualització tals com en que mesura difondre aquesta actualització, amb quina freqüència enviar aquest missatge d'actualització, etc; s'han de deixar pels algorismes de *peer*, més que a dins de l'abast del protocol.

La informació d'estat de *peer* ha de poder-se anunciar entre els *peers* mitjançant els missatges d'actualització o report de l'estat del *peer*.

Els *peers* han d'implementar el *Peer Protocol* per peticions i respostes de *chunks* entre els *peers* abans de que el contingut *streaming* es comenci a transmetre.

3.2.4. Requeriments de protecció de sobrecàrrega i gestió d'errors

A més dels requeriments citats anteriorment, el protocol hauria de complir uns certs requeriments de protecció de sobrecàrrega i gestió d'errors. Entre aquests, un *peer* ha de ser capaç de respondre amb informació d'error responent als altres *peers* enviant missatges PPSP, quan alguna informació, com pot ser la llista de *peers* o l'expressió dels *chunks*, no es pot entendre a dins del missatge de petició original.

Fig. 3.1.- Control d'errors

Centrant-nos ara a la part del *Tracker* del PPSP, els requeriments tenint en compte que opera de forma pròxima a la seva capacitat límit, han de ser:

- El *Tracker* de PPSP ha de permetre informar als *peers* sobre la seva situació de sobrecàrrega i, per tant, redirigir-los cap a un altre *Tracker* de PPSP que estigui disponible.

- El *Tracker* de PPSP ha de permetre informar als *peers* sobre la seva situació de sobrecàrrega i, per tant, fer-los finalitzar la conversació amb el *Tracker* de PPSP actual.

- El *Tracker* de PPSP ha de permetre informar als *peers* sobre la seva situació de sobrecàrrega i, per tant, evitar nous intents de converses per part d'altres *peers* que puguin estar interessats.

3.3. Consideracions de seguretat

El que es pretén amb aquest apartat és analitzar de forma genèrica les diferents amenaces de seguretat possibles i per tal proporcionar uns requeriments que serveixin com a consideracions de seguretat.

El primer dels requeriments de seguretat és que PPSP ha de suportar eixams tancats, on els *peers* estaran autentificats. Això assegurarà que només els usuaris autentificats poden accedir al contingut mèdia original al sistema d'*streaming* P2P. Això es pot aconseguir mitjançant mecanismes de seguretat tals com l'autentificació d'usuari i/o l'esquema de gestió de claus.

D'altra banda, la confidencialitat del contingut *streaming* a PPSP ha de ser suportada i l'esquema corresponent de gestió de claus ha d'escalar bé al sistema *streaming* P2P. A més, PPSP ha de proporcionar una opció per tal de xifrar l'intercanvi de dades sobre les diferents entitats de PPSP.

PPSP ha de tenir mecanismes per limitar els danys potencials causats pel malfuncionament i mal comportament dels *peers* a un sistema *streaming* P2P. Un atac d'aquest tipus degradarà la qualitat del mèdia prestats al receptor. Per exemple, a un sistema d'*streaming* de vídeo en viu P2P un usuari contaminant pot introduir *chunks* corruptes. Cada receptor integra a dins del seu flux de reproducció els *chunks* contaminats que rep dels seus altres veïns. Donat que els *peers* envien *chunks* cap a altres *peers*, el contingut contaminat pot estendre's potencialment a través de molta part de la xarxa d'*streaming* P2P. Per tot això, PPSP ha de donar suport a l'identificació dels *peers* amb mal comportament, i excloure'ls o expulsar-los del sistema d'*streaming* P2P.

PPSP haurà de prevenir als *peers* d'atacs de tipus DoS que esgota els recursos disponibles d'un sistema d'*streaming* P2P. Donada la prevalença d'atacs d'aquest tipus, els de negació de servei, a Internet, és important adonar-se de que una amenaça semblant pot existir a un sistema *streaming* a gran escala on els atacants són capaços de consumir moltíssims recursos amb un petit esforç.

PPSP haurà de ser robust, és a dir, quan un *Tracker* centralitzat falla, el sistema d'*streaming* P2P ha de seguir funcionant mitjançant el suport de *Trackers* distribuïts.

Es pretén també que els mecanismes de seguretat de P2P es pugui usar a la mesura del possible a PPSP, per tal d'evitar el desenvolupament de nous mecanismes de seguretat.

4. PPSP Tracker Protocol

4.1. Introducció

PPSP està composat de dos protocols: *PPSP Tracker Protocol* i *PPSP Peer Protocol*. El primer d'ells, PPSP *Tracker* Protocol, proporciona comunicació entre *Trackers* i *Peers*, de forma que els *peers* informen de l'estat de l'*streaming* al *Tracker* i demanen llistes de candidats del *Tracker*.

El que es presenta aquí és una proposta pel *PPSP Tracker Protocol*, que encara no s'ha definit i acceptat oficialment sinó que està en desenvolupament, basant-se en les necessitats i requeriments sorgits a les discussions generades a la llista de correus oficial del IETF. Fent, primerament, un anàlisi de les entitats funcionals que incloses al *Tracker Protocol*, així com una llista de funcions i requeriments d'alt nivell. Principalment és fa una descripció abstracta de les operacions requerides pel *PPSP Tracker Protocol*, així com les definicions formals de sintaxi, semàntica i instruccions de processat de missatge detallades pel *PPSP Peer Protocol*.

Com ja hem vist, la majoria de protocols P2P implementats a l'actualitat són propietaris, de forma que el que es pretén és extreure les característiques fonamentals, funcionalitats i politiques dels protocols propietaris, per així estendre'ls basant-se en l'experiència d'implementació.

Aquest *PPSP Tracker Protocol* és un protocol del nivell d'aplicació per *peers* per tal de registrar, publicar/demanar contingut i proporcionar l'estat del *peer* a un *Tracker*. També s'usa pels *Trackers* per proporcionar llistes de *peers* als *peers*, així com per enviar missatges de control i de gestió a altres *Trackers* i per comunicar-se amb aquests. El *PPSP Tracker Protocol* pot oferir aplicacions de *live media* i VoD, així com aplicacions per compartir fitxers.

Pel que fa a la seguretat del protocol, aquest és un tema que no està cobert encara. Es pot anticipar que s'usarà un esquema certificat públic però encara no hi ha res implementat. Es deixarà pendent de cara a la implementació final del protocol.

4.1.1. Entitats funcionals

Existeixen dos tipus diferenciats de nodes, o entitats funcionals, involucrades al *Tracker* Protocol dels sistemes *streaming* P2P: *Trackers* i *peers*. Els *peers* són nodes que estan activament enviant i rebent contingut mèdia en *streaming* i inclouen tant hosts connectats estàticament com dispositius mòbils amb connectivitat i adreces IP que

canviïn amb el temps. El conjunt de *peers* que participen a la sessió d'*streaming* anirà variant dinàmicament en funció del temps. El *Trackers* són nodes coneguts amb connectivitat estable que manté meta-informació sobre el contingut en *streaming* i el conjunt dinàmic de *peers*. Els *Trackers* és poden organitzar de forma centralitzada o distribuïda.

El *Tracker* és una entitat lògica que s'encarrega d'emmagatzemar informació sobre els *peers*, concretament quines parts de la informació, o *chunks*, poden proporcionar cadascun dels *peers* que hi ha a la xarxa. El *Tracker* també emmagatzema l'estat en el que es troben els *peers*, de forma que aquesta informació ajuda al *Tracker* a triar el candidat apropiat per a un *peer* demanant, així com també pot emmagatzemar una llista de quins *chunks* disposa cada *peer*.

D'altra banda, els *peers* són els dispositius que participen activament a la compartició d'informació relacionada a un fitxer o *stream* en particular. Cadascun dels *peers* emmagatzema algunes parts, anomenades *chunks*, i contacta amb el *Tracker* per avisar sobre quina informació té disponible. Quan un *peer* desitja obtenir informació, contacta amb el *Tracker* per trobar altres *peers* que estiguin participant a l'eixam, i d'aquesta forma obtenir una llista dels *peers* amb els *chunks* que poden proporcionar. És a dir, els *peers* es comuniquen entre ells per intercanviar *chunks* i, en el cas en el que el *Tracker* emmagatzema només la llista de *peers* de l'eixam, per intercanviar llistes de *chunks*.

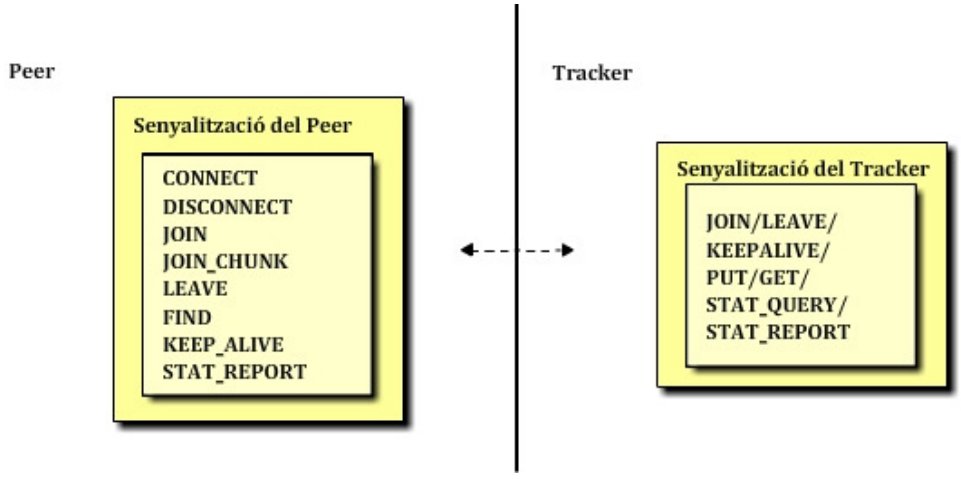

Fig. 4.1.- Components del Tracker Protocol

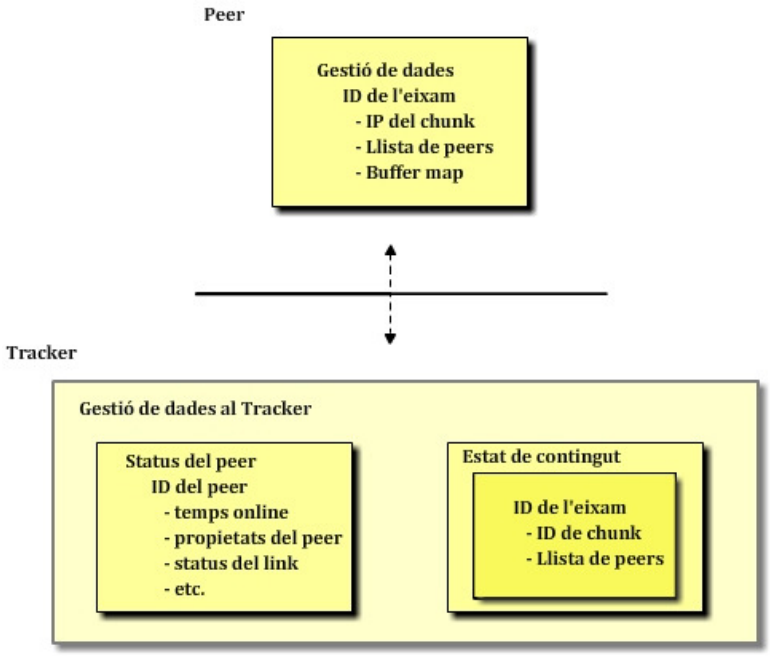

Fig. 4.2.- Components de gestió de dades a PPSP

4.1.2. Assumpcions

Quan un *peer* desitja unir-se a una aplicació existent d'*streaming* P2P primer ha de localitzar a un *Tracker*. Els *peers* poden usar qualsevol mètode per trobar un *Tracker*, com per exemple obtenir-lo de mecanismes de aprovisionament d'un proveïdor de serveis, d'una pàgina web o via *broadcast*.

De forma similar, s'ha d'assumir que els *peers* han obtingut els seus *Peer ID* i qualsevol certificat requerit per seguretat amb mecanismes *out-of-band*. Concretament, i ja ho veurem més endavant quan es parli de l'aplicació que acompanya al projecte, li donarem nosaltres via un fitxer de configuració inicial. Això es déu a que aquesta funcionalitat no s'ha discutit encara de forma oficial.

4.1.3. Descripció

PPSP Tracker Protocol és un protocol del tipus petició-resposta. És a dir, les peticions s'envien i llavors unes respostes retornen contestant a aquestes peticions. Tot i que la majoria de peticions s'envien al *Tracker* des de *peers* que sol·liciten informació, el *Tracker* també pot enviar missatges als *peers* per demanar informació.

Cal comentar que el *Tracker Protocol* no s'encarrega d'intercanviar cap tipus de dades entre els *peers*, és a dir, no té assignades labors de transport. En comptes d'això, el *Tracker* manté una llista de *peers* que participen en un determinat *stream* o tenint *chunks* d'una sessió en particular, i és aquesta informació la que comparteix.

4.1.3.1. Tipus de codificació

Hi han dues opcions per presentar el protocol segons el tipus de codificació: la codificació en llenguatge ASCII, basada en una codificació amb cos d'XML portat a sobre d'HTML; i la codificació binaria.

La primera d'elles és més senzilla d'entendre en general, però presenta l'inconvenient de que necessita més espai i ample de banda. D'altra banda, la segona de les dues codificacions, és molt més lleugera tant en el sentit d'ample de banda com de processament, però presenta major dificultat a l'hora d'entendre-la.

Considerant que els *peers* es comuniquen amb molta freqüència i que les aplicacions d'*streaming* són molt sensibles als retards, la codificació binaria del *PPSP Tracker Protocol* per l'enviament de missatges serà més òptima en aquest sentit.

Deixant de banda quina de les dues codificacions s'utilitza, hi ha un seguit de mètodes o tipus de missatges per cadascuna de les accions que ha realitzar el *Tracker* Protocol i que comparteixen les dues codificacions.

4.1.3.2. Mètodes

PPSP Tracker Protocol és un protocol del tipus petició-resposta. Aquestes peticions i respostes s'envien en forma de missatges on cadascun d'aquests missatges té unes característiques pròpies segons si és de petició o de resposta i, dins d'ambdues diferenciacions, de quin tipus de petició o de resposta és tracta.

Aquests tipus diferenciats de missatges és el que es coneix com a mètodes, i es veuen amb detall als seus respectius apartats: *5.3. Missatges de petició* i *5.4. Missatges de resposta*. A continuació hi ha unes pinzellades dels diferents mètodes dels missatges de petició:

- **CONNECT:** És el primer missatge que ha d'enviar un *peer*. Abans d'això, no estarà connectat amb el *Tracker*. Al connectar, el *Tracker* obté informació del *peer* que fa la petició i li respon com correspongui. A partir d'aquí, es pot procedir amb la resta de missatges de petició.

- **DISCONNECT:** És el missatge que serveix per què el *peer* abandoni completament el sistema i no continuï tenint connexió amb el *Tracker*. El *Tracker* esborrarà tota la informació del *peer* un cop procedida la petició i la resta de *peers* no podran accedir-hi més.

- **JOIN i JOIN_CHUNK:** Els *peers* usen un missatge JOIN per notificar al *Tracker* el desig d'entrar a un eixam en particular, però sense indicar de quins *chunks* disposa. JOIN_CHUNK fa la mateixa funció, però indicant els *chunks*.

- **LEAVE:** Els *peers* usen un missatge d'aquest tipus quan volen sortir d'un eixam en particular i no de tot el sistema. El *Tracker* esborrarà la informació d'aquest *peer* a l'eixam corresponent i no la proporcionarà a altres *peers*.

- **FIND:** Aquest és un mètode pel qual el *peer* demana al *Tracker* una llista de *peers* que puguin oferir un contingut concret o que estiguin en un eixam determinat.

- **KEEPALIVE:** Missatges que s'envien de forma periòdica dels *peers* al *Tracker* per notificar que el *peer* encara està viu. Si el *Tracker* no rep aquest missatge durant un temps preconfigurat, assumeix que el *peer* ha mort i l'esborra.

- **STAT_QUERY:** Mètode que funciona en dues direccions. De *peer* a *Tracker* proporciona informació estadística als *peers* sobre el sistema. De *Tracker* a *peer* proporciona informació sobre el *peer* en particular.

- **STAT_REPORT:** Mètode que funciona en dues direccions. De *peer* a *Tracker* permet als *peers* enviar dades estadístiques al *Tracker*. De *Tracker* a *peer* serveix per notificar al *peer* la informació general de tot el sistema.

Aquests eren els diferents mètodes dels missatges de petició. Per a respondre a aquestes peticions, com ja s'ha comentat, el *Tracker* genera tot un seguit de missatges de resposta que variaran en funció dels paràmetres i mètode rebuts i de diverses característiques internes del *Tracker*. Els diferents mètodes dels missatges de resposta són els següents:

- **SUCCESFUL:** Missatge que serveix com a indicació de que la petició rebuda ha sigut acceptada i processada correctament. Segons diferents factors, el missatge SUCCESFUL pot variar i contenir dades al cos de missatge. Això es veurà amb més detall

- **INVALID_SYNTAX:** Missatge que indica que hi ha un error al format del missatge de petició que s'ha rebut.

- **VERSION_NOT_SUPPORTED:** Missatge per indicar que la versió utilitzada del protocol o els cossos de missatge, no són correctes.

- **MESSAGE_NOT_SUPPORTED:** Missatge que serveix per indicar que el missatge de petició rebut no està suportat pel *Tracker*. Com a exemple, alguns *Trackers* poden no implementar les funcions estadístiques.

- **TEMPORARILY_OVERLOADED:** Missatge per indicar que el servidor està temporalment sobrecarregat.

- **INTERNAL_ERROR:** Missatge per indicar que el servidor no està disponible per processar la petició degut a un error intern.

- **MESSAGE_FORBIDDEN:** Missatge per indicar que al *peer* que fa la petició no li està permès a fer aquesta petició.

- **OBJECT_NOT_FOUND:** Missatge per indicar que l'objecte demanat, ja sigui un eixam o un *chunk*, no s'ha pogut trobar.

- **AUTHENTICATION_REQUIRED:** Aquest missatge de resposta serveix per indicar que es requereix autenticació per accedir a aquest contingut del *Tracker*.

- **PAYMENT_REQUIRED:** Missatge per indicar que el contingut al que es vol accedir és de pagament.

4.2. Codificació binaria dels missatges

Considerant que els *peers* el més segur és que es comunicaran amb el *Tracker* de forma freqüent, i que les aplicacions d'*streaming* són força sensibles als retards, una codificació binaria en forma de trames s'ha de tenir present a l'hora de dissenyar el *PPSP Tracker Protocol*.

A dins d'aquest apartat, es tindrà en compte aquesta consideració i es detallarà una definició de la normativa de la codificació binària en trames dels diferents conjunts de missatges que engloben el *PPSP Tracker Protocol*, tant missatges de petició com de resposta.

Els missatges que formen el *Tracker Protocol* estan dividits en dues parts. La primera d'aquestes és una part més genèrica, compartida per tots els tipus de missatges, anomenada *Message Header*; i la segona part, que és diferent per cada tipus de missatge, anomenada *Message Bodies*, que no té un format fix sinó que varia en funció del tipus de missatge.

4.2.1. Capçalera dels missatges

La capçalera general dels missatges, anomenada també capçalera compartida, que intercanvia el *PPSP Tracker Protocol* és de la següent forma:

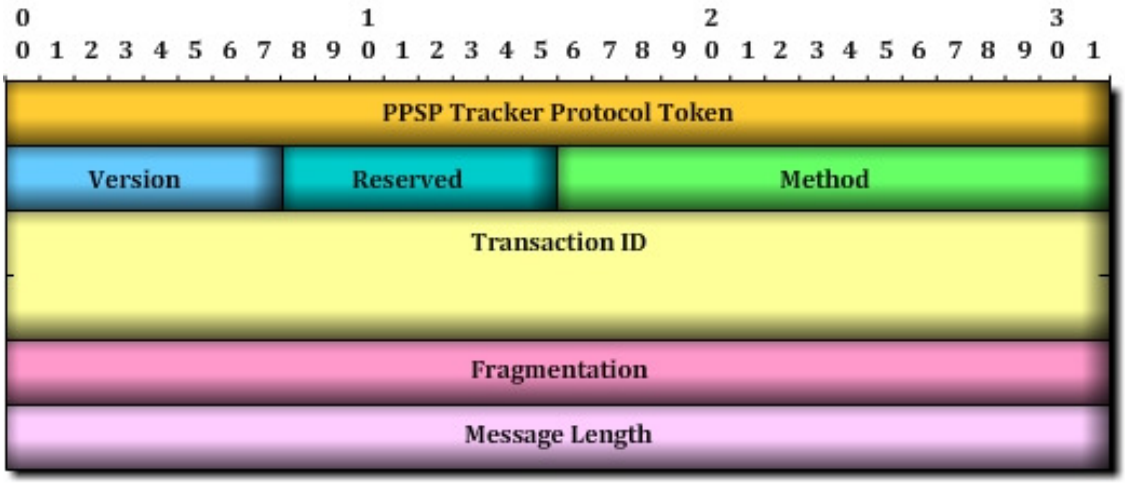

Fig. 4.3.- Capçalera general de missatge de PPSP Tracker Protocol

Com es pot observar, la capçalera la conformen diversos camps que serà informació comú a tots els missatges. Aquests camps que conformen la capçalera són:

- **PPSP Tracker Protocol Token**: És un camp fix de 32 bits que indica al receptor que aquest es un missatge corresponent al *PPSP Tracker Protocol*. El valor que ha de tenir sempre es correspon a 0xd0505374, o el que és el mateix, la cadena "PPSt" amb el bit d'ordre més alt del primer byte activat. En cas de que el *Peer* Protocol usi la mateixa codificació, es podria posar el valor més genèric, com per exemple a "PPSP".

- **Version**: Camp de 8 bits que indicarà la versió del protocol que s'està usant. La versió serà un valor fix ubicat entre 0.1 i 25.4. A la versió a la que es troba el protocol actualment, el camp *Version* ha de ser posat forçosament a 0.1. En una primera versió del PPSP *Tracker* Protocol, aquest camp estava fixat a 0.0 i la divisió dels 8 bits es feia de forma que els 4 de més pes formaven part del nombre a l'esquerra del punt i els altres 4 bits de la part a la dreta del punt. Degut a que la nova versió del protocol argumenta que el valor de versió màxim pot ser fins a 25.4, i el nombre màxim que es pot representar amb 4 bits és el 15, he canviat la forma en que es reparteixen els bits fent que els 5 de major pes corresponguin al primer valor i els altres 3, al segon.

- **Reserved**: 8 bits que estan reservats, com bé diu el seu nom, per a usos especials, com podria ser per estendre el *namespace* quan aquest estigui exhaust. Els valors del camp *Reserved* no es contemplen encara.

- **Method**: Camp de 16 bits que serveis per indicar el tipus de mètode del missatge. És el camp que indica de quin tipus serà el *Message Body* que hi haurà a continuació de la capçalera del missatge. Els diferents valors que pot adoptar el camp *Method* es poden veure a les següents taules, separades segons si són missatges de petició o missatges de resposta:

Nom del Method de la petició	Valor del camp Method
CONNECT	0x0800
DISCONNECT	0x0801
JOIN	0x0802
JOIN_CHUNK	0x0803
LEAVE	0x0804
FIND	0x0805
KEEPALIVE	0x0806
STAT_QUERY	0x0807
STAT_REPORT	0x0808

Fig. 4.4.- Codificació del camp Method per missatges de petició

Nom del Method de la resposta	Valor del camp Method
SUCCESFUL (OK)	0x8800
INVALID SYNTAX	0X88F0
VERSION NOT SUPPORTED	0x88F1
MESSAGE NOT SUPPORTED	0x88F2
TEMPORARILY OVERLOADED	0x88F3
INTERNAL ERROR	0x88F4
MESSAGE FORBIDDEN	0x8810
OBJECT NOT FOUND	0x8811
AUTHENTICATION	0x8812
PAYMENT REQUIRED	0x8813

Fig. 4.5.- Codificació del camp Method per missatges de resposta

- **Transaction ID**: El conforma un camp de 64 bits que identifica la transacció i també permet als receptors que discerneixin transaccions que d'altra forma serien idèntiques. Les respostes usen el mateix *Transaction ID* que la petició a la que corresponen. A més a més, el camp *Transaction ID* també s'utilitza per a rejuntar fragments.

- **Fragmentation:** Camp format per 32 bits que indica si el missatge s'ha fragmentat en varies parts, degut a la seva longitud, o no. La fragmentació s'explica amb detall en un apartat posterior.

- **Message Length:** Aquest camp indica la longitud del missatge en bytes, incloent la pròpia capçalera. És un camp de 32 bits. En el cas de que un missatge estigui fragmentat, el camp *Message Length* indica la mida de cadascun dels missatges per separat, concretament el que li correspongui, i no la suma total de la mida dels fragments.

4.2.2. Cos dels missatges

El cos dels missatges, a diferència de la capçalera, no és comú a tots els tipus de missatge, sinó que és el tret diferencial que distingeix uns missatges de altres. És el que s'anomena *Message Bodies*, i no té un format fix sinó que varia en funció del tipus de missatge. Tot i ser cadascun diferent, podem agrupar els missatges segons si són missatges de petició o missatges de resposta a aquestes peticions.

4.2.3. Fragmentació

Actualment, no hi ha retransmissió fragment a fragment, i no hi ha forma de saber d'avançat quantes parts hi ha, tot i que l'offset permet la detecció de fragments perduts. Addicionalment, si aquests missatges no s'usen pel tràfic dels *peers*, *Path MTU (PMTU) discovery* extrem a extrem tindria sentit per determinar la longitud total dels fragments. De forma similar, necessitem decidir un temps de retransmissió òptim per aquest protocol. Finalment, amb aquesta aproximació, els missatges estan efectivament limitats a 2^24 + (PMTU – 32), el que podria ser insuficient.

Ja que els missatges transmesos pel PPSP *Tracker* Protocol poden ser molt grans, i poden ser transportats sobre mitjans de transport poc fiables, es necessari un mecanisme de fragmentació robust. S'ha optat per reutilitzar el mecanisme descrit al document [I-D.ietf-p2psip-base]. Porcions de la descripció processada en aquesta secció s'han agafat prestades d'aquest document.

Qualsevol missatge enviat pot ser fragmentat. Cada missatge que ha sigut fragmentat ha d'incloure una copia de la capçalera general de missatge. Quan un missatge s'ha de fragmentar, s'ha de dividir en fragments d'igual mida que no poden ser més llargs que el PMTU del següent enllaç de xarxa menys 32.

Després de la recepció d'un missatge transmès del *peer* previst, el *peer* manté els fragments a un *buffer* fins que el missatge sencer es rebut. Llavors el missatge és re-ensamblat en un de sol i processat. A fi de mitigar els atacs de denegació de servei, els receptors han de finalitzar el temps d'espera amb un *timeout*, els fragments incomplets després d'un temps de vida de petició màxim, que serà de 15 segons. Aquest *timeout* s'ha agafat prestat de RELOAD. Si el receptor acaba l'espai del *buffer* per re-ensamblar els missatges, haurà de deixar el missatge.

Quan un missatge està fragmentat, el valor de l'offset del fragment s'emmagatzema als 24 bits més baixos del camp de fragment a la capçalera general de missatge. El offset és el nombre de bytes entre el final de la capçalera i l'inici de transmissió de dades. El primer fragment té per tant un offset de 0. Els indicadors del primer i últim bit han d'estar fixats adequadament. Si el missatge no està fragmentat, llavors tant com el primer com l'últim fragment es fixen a 1 i l'offset és 0, donant com a resultat un valor de fragment de 0xC0000000. Després d'un complet ensamblament del missatge, el processament es fa com normalment.

4.3. Missatges de petició

PPSP Tracker Protocol és un protocol del tipus petició-resposta. És a dir, les peticions s'envien i llavors unes respostes retornen contestant a aquestes peticions. Tot i que la majoria de peticions s'envien al *Tracker* des de *peers* que sol·liciten informació, el *Tracker* també pot enviar missatges als *peers* per demanar informació.

Cal comentar que el *Tracker Protocol* no s'encarrega d'intercanviar cap tipus de dades entre els *peers*, és a dir, no té assignades labors de transport. En comptes d'això, el *Tracker* manté una llista de *peers* que participen en un determinat *stream* o tenint *chunks* d'una sessió en particular, i és aquesta informació la que comparteix.

A continuació es fa una descripció detallada d'aquests diferents missatges de petició, així com del seu ús i estructura, ampliant el que ja s'havia avançat prèviament. Són el que es coneix com *PPSP Tracker Protocol Methods* i són les dades addicionals que s'afegeixen després de la capçalera.

4.3.1. CONNECT Message

El missatge de tipus CONNECT s'envia quan un *peer* desitja unir-se a un sistema i connectar amb el *Tracker*. El *peer* ha d'haver adquirit prèviament un *Peer ID* via qualsevol mecanisme *out-of-band*.

El procediment de connexió segueix dos passos. Al primer d'ells, el *peer* envia un missatge de CONNECT cap al *Tracker*, indicant la seva intenció de voler connectar. Tot seguit, el *Tracker* generarà una resposta indicant l'èxit, fracàs o qualsevol altra condició que déu ser satisfeta per procedir a la connexió amb èxit, com podria ser acreditar-se.

Per a formar un missatge de tipus CONNECT, el *peer* que fa l'enviament construeix un missatge general amb la seva capçalera, col·locant correctament tots els camps d'aquesta, posant el camp *Method Field* a CONNECT, que ja hem vist que correspon al missatge codificat 0x0800, i generant de forma aleatòria un *Transaction ID*. El cos del missatge serà forçosament el *Peer ID* del *peer*, conformant aquest camp un camp de 160 bits.

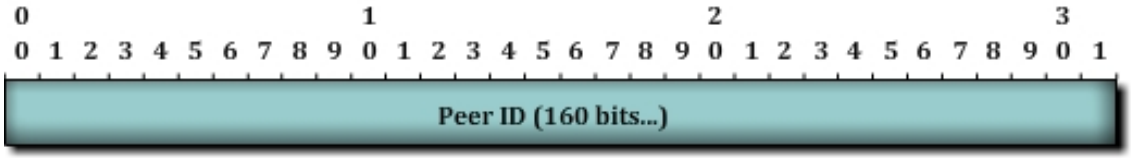

Fig. 4.6.- Cos del missatge CONNECT

4.3.2. DISCONNECT Message

En contrapunt al missatge de CONNECT, existeix el missatge DISCONNECT, que s'envia quan un *peer* desitja deixar d'estar unit a un sistema i desconnectar del *Tracker*.

Per a formar un missatge DISCONNECT, el *peer* que fa la petició construeix la capçalera de missatge general. S'han de posar correctament tots els camps, fent èmfasi als camps *PPSP Tracker Protocol Token* i *Version*. A dins del camp *Method*, s'indicarà que es un missatge de DISCONNECT, amb codificació equivalent a 0x0801, i *Transaction ID* s'haurà generat aleatòriament de forma prèvia. La longitud total del missatge equivaldrà als 192 bits de la capçalera més 160 bits del cos de missatge, que inclourà només el *Peer ID* del *peer*. Això indicarà al *Tracker* que el *peer* s'està desconnectant i implícitament informa al *Tracker* de que el *peer* ja no estarà disponible a cap dels eixams.

És a dir, un cop rebut un missatge de DISCONNECT, el *Tracker* ha d'esborrar el *peer* corresponent de la llista de *peers*, que no podrà fer cap mena de peticions fins que torni a connectar. El *Tracker* ha d'esborrar al *peer* de tots els eixams de la mateixa forma que si hagués rebut un missatge de tipus LEAVE per cadascun dels eixams als que el *peer* participa per separat.

Un tema que encara està obert a debat, és que com a conseqüència d'aquest disseny, serà necessari tenir un mapatge de tots els eixams als que el *peer* està unit, de forma que es pugui esborrar de tots al fer un DISCONNECT.

La forma del cos de missatge, que inclou un únic camp anomenat *Peer ID*, serà de la forma següent:

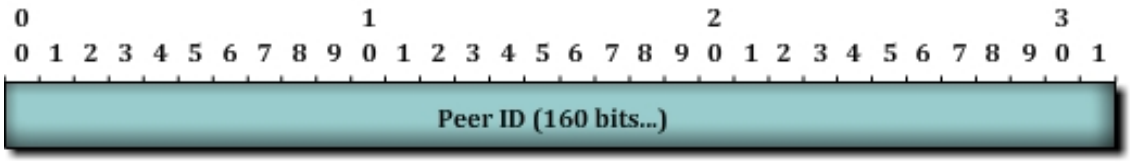

Fig. 4.7.- Cos del missatge DISCONNECT

4.3.3. JOIN Message

Els missatges de tipus JOIN, tant JOIN com JOIN_CHUNK, són missatges usats pel *peer* amb la intenció de informar al *Tracker* que voldria participar o unir-se a un eixam en particular i, opcionalment, quines porcions d'aquest eixam té disponibles. Degut a que els missatges de tipus JOIN i JOIN_CHUNK, tot i semblar que són el mateix, estan ben diferenciats, per aquest motiu es divideix l'apartat en dos, un per cadascun dels missatges.

Els missatges de tipus JOIN s'utilitzen quan el *peer* té alguns *chunks* o desitja unir-se a un *stream* en viu, però no proporciona la llista de *chunks* o la localització de l'*stream* al *Tracker*. És a dir, la conversa s'usa entre els *peers* per intercanviar la llista de *chunks* o la posició de l'*stream*, o simplement s'uneix a la posició actual. Cal comentar que actualment no existeix encara un mecanisme per a representar una posició a un *stream*, però si que es poden codificar *chunks*.

Per a formar un missatge de tipus JOIN, cal construir la capçalera general de missatge i omplir-la amb els camps corresponents. Al camp *Method*, s'indicarà que és un missatge de tipus JOIN, que correspon a la seqüència codificada 0x0802. La resta de camps de la capçalera s'omplen amb els valors habituals. D'altra banda, el cos del missatge, el conformen tres camps diferenciats.

- **Peer ID:** Camp de 160 bits que indicarà l'identificador del *peer* que envia el missatge. Aquest *Peer ID* l'ha d'haver adquirit el *peer* prèviament via qualsevol mecanisme *out-of-band*.

- **Swarm ID:** Camp de 128 bits que serveix per indicar l'identificador de l'eixam del que el *peer* està interessat.

- **Expiration Time:** Camp de 32 bits. Aquest temps d'expiració haurà de valer algun valor diferent de zero, expressat en segons, que indicarà que el *peer* espera no seguir participant a l'eixam al final del temps. Si per contra, el *peer* no desitja establir un temps d'expiració, el valor del camp haurà de ser zero.

La resposta que donarà a aquest tipus de missatges tindrà el mateix *Transaction ID* que la petició i no contindrà cap tipus d'informació addicional al cos del missatge, tan sols la requerida per la capçalera, indicant al *Method* el tipus de missatge que serà la resposta. El cos de missatge de tipus JOIN, amb els seus camps, es pot veure a la figura següent:

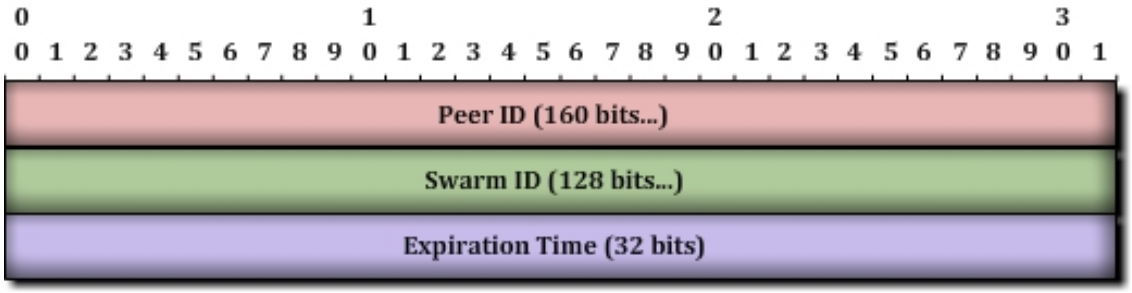

Fig. 4.8.- Cos del missatge JOIN

4.3.4. JOIN_CHUNK Message

Continuant amb l'apartat anterior, JOIN message, on ha quedat pendent el segon dels missatges d'aquest tipus: JOIN_CHUNK message. Els missatges de tipus JOIN_CHUNK, a diferència dels de tipus JOIN, s'utilitzen quan el *peer* desitja proporcionar una llista detallada de *chunks* o una posició a l'*stream* cap al *Tracker*.

Per a formar un missatge de tipus JOIN_CHUNK, el procediment a seguir és exactament igual al que s'ha de seguir per tal de construir un missatge de tipus JOIN, estant la única diferencia al camp *Method* de la capçalera, que indicarà que és un missatge de tipus JOIN_CHUNK (tindrà el valor 0x0803) i al cos del missatge, que a més dels camps que incorpora un JOIN *Message*: *Peer ID*, *Swarm ID* i *Expiration Time*; té afegits uns camps nous que els segueixen amb funcions diverses:

- **Nombre de Chunks:** Camp de 16 bits que serveix per indicar el nombre de *chunks* que hi ha a continuació, no la longitud de la llista de *chunks*.

- **Reserved:** Camp de 16 bits que encara no té un ús definit. Es reserva per futurs usos.

- **Chunk ID:** Camp de 30 bits que serveix per indicar l'identificador del Chunk. El camp va de la ma amb el petit camp següent de 2 bits, de forma que es pot considerar que el ID de cada *chunk* es de 32 bits realment.

- **S:** Camp de 2 bits que serveix per indicar continuació, usat com a notació compacta per indicar una seqüència de *chunks*. Si el *Chunk ID* no representa una sèrie, sinó un *chunk* individual, aquest camp s'ha de forçar a 00. Si el *Chunk ID* és l'inici d'una sèrie, aquest camp haurà de valer 01, i si és el final d'una sèrie, el seu valor serà 10. El mètode d'assignació d'aquests valors es deixa a la implementació. Les sèries s'han d'especificar en parells, és a dir, una sèrie consistirà en dos *chunks*, el primer amb 01 i el segon amb 10. Les sèries han de ser continues i creixents.

- **Chunk ID addicionals:** És una representació per indicar que hi poden haver més d'un Chunk ID + S. Tants com indiqui el camp *Nombre de Chunks*. Podem considerar llavors que aquest "camp" equival a un múltiple de 32 bits.

Quan un missatge de tipus JOIN o JOIN_CHUNK es rebut, el *Tracker* processarà la petició. Si el missatge es acceptat pel *Tracker*, aquest ha de verificar que els camps estan correctament formats i l'ha de rebutjar amb un *INVALID SYNTAX response* si el missatge està mal format. En cas de no ser així, el *Tracker* l'acceptarà i entrarà la informació a l'emmagatzemament de dades intern, responent amb un *SUCCESFUL message response*. La resposta que donarà a aquest tipus de missatges tindrà el mateix *Transaction ID* que la petició i no contindrà cap tipus d'informació addicional al cos del missatge, només la requerida per la capçalera, indicant al *Method* el tipus de missatge que serà la resposta.

En una connexió en temps real, el *peer* i el *Tracker* estaran ocupats processant JOIN_CHUNK *messages* i respostes. Si la presentació d'informes ha de ser periòdica, aquest període s'ha de configurar amb cura, o el *peer* podria no operar de forma eficient. Una alternativa seria que el *Tracker* comuniqués de forma dinàmica la disponibilitat dels *chunks* dels *peers* candidats. A un eixam de VoD, un *peer* continuarà rebent *chunks* continus d'un eixam designat. Després de que un faci un JOIN_CHUNK a un primer *chunk*, el *Tracker* podrà comunicar quins *chunks* estan actualment emmagatzemats al *peer*.

A la figura següent es pot veure el cos de missatge de tipus JOIN_CHUNK:

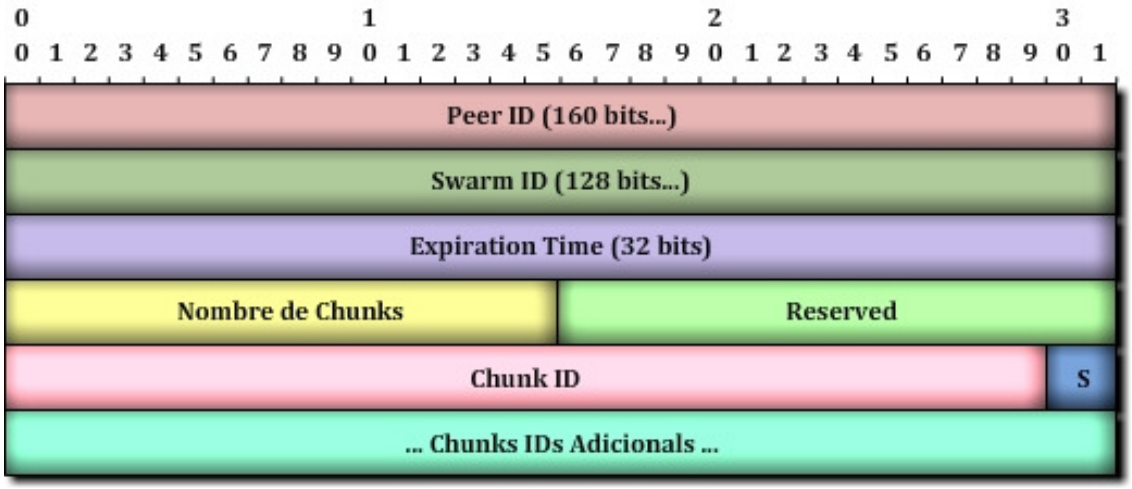

Fig. 4.9.- Cos del missatge JOIN_CHUNK

4.3.5. LEAVE Message

Els missatges de tipus LEAVE s'utilitzen pel *peer* amb la intenció d'informar al *Tracker* que ja no desitja participar en un eixam en particular. El podríem definir com una versió de DISCONNECT per eixams en particular.

Un missatge de tipus LEAVE es forma igual que tots els missatges anteriors. Consta d'una capçalera de missatge que s'ha d'omplir amb diferents valors per cada camp i d'un cos de missatge. Al camp *Method* de la capçalera s'ha de posar que es tracta d'un missatge de tipus LEAVE, codificant amb el valor 0x0804.

El cos de missatge, que és la part diferencial de tots els tipus de missatges, consta en aquest cas de dos camps:

- **Peer ID:** Camp de 160 bits que indicarà l'identificador del *peer* que envia el missatge. Aquest *Peer ID* l'ha d'haver adquirit el *peer* prèviament via qualsevol mecanisme *out-of-band*.

- **Swarm ID:** Camp de 128 bits que indica l'identificador de l'eixam del que el *Peer* vol deixar de formar part.

Un cop el *Tracker* ha rebut un missatge de tipus LEAVE, es processarà la petició, i segons sigui correcta o no, l'acció a prendre i la resposta que es donarà, serà diferent. En cas de que el missatge de petició hagi sigut correcte, sense cap mena d'error de sintaxi, el *Tracker* respondrà amb un missatge SUCCESFUL i esborrarà al *peer* de la llista de *peers* participants a un eixam particular. El *Tracker* no notificarà a altres *peers* de que aquest *peer* ha abandonat l'eixam, aquesta funcionalitat es deixa al *Peer* Protocol.

La resposta tindrà el mateix *Transaction ID* que la petició i no contindrà cap informació addicional al cos de missatge.

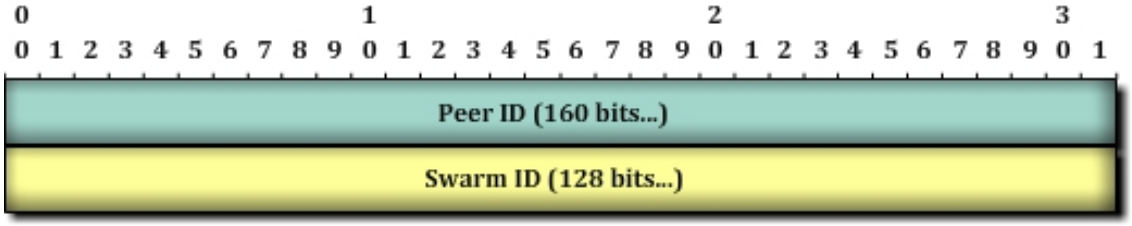

Fig. 4.10.- Cos del missatge LEAVE

4.3.6. FIND Message

Els *peers* usen el mètode FIND per demanar al *Tracker* que els hi retorni llistes de *peers* que poden proporcionar un contingut específic o que estiguin en un eixam en particular. Al rebre un missatge de tipus FIND, el *Tracker* troba els *peers* candidats llistats al *content status* i retorna la llista al *peer* que l'ha demanat. El *Tracker* pot prendre l'estat del *peer* i la prioritat dels *peers* en consideració quan escull un *peer* candidat per afegir a la seva llista. La prioritat dels *peers* es pot determinar per la preferència de topologia de la xarxa, entre d'altres.

En *live streaming*, quan es rep un missatge FIND, el *Tracker* també actualitza el content status per involucrar el nou *peer* sota un canal específic. L'operació que desenvolupa el missatge FIND pot especificar una localització particular a dins d'un *stream* i/o *chunk* en el que el *peer* estigui interessat.

A l'hora de generar un missatge de tipus FIND, es construeix la capçalera general de missatge, indicant al camp *Method* que és un missatge de tipus FIND, al que li correspon la codificació 0x0805. El cos de missatge presenta tres camps:

- **Peer ID:** Camp de 160 bits que indica l'identificador del *peer* que fa la petició de missatge i que ha aconseguit prèviament amb qualsevol mecanisme.

- **Swarm ID:** Camp de 128 que serveix per identificar l'eixam del que el *peer* està interessat en obtenir *chunks*.

- **Chunk ID:** Camp de 32 bits que ha de estar fixat a un valor particular de *chunk* en cas de que el *peer* estigui interessat en *chunks* que tinguin aquest valor de Chunk ID o majors, ja que el *Tracker* només retornaria *peers* que tinguin *chunks* amb aquest identificador o amb un valor major. En cas de que el *peer* estigui interessat en qualsevol *chunk*, el valor s'ha de fixar tot a zero.

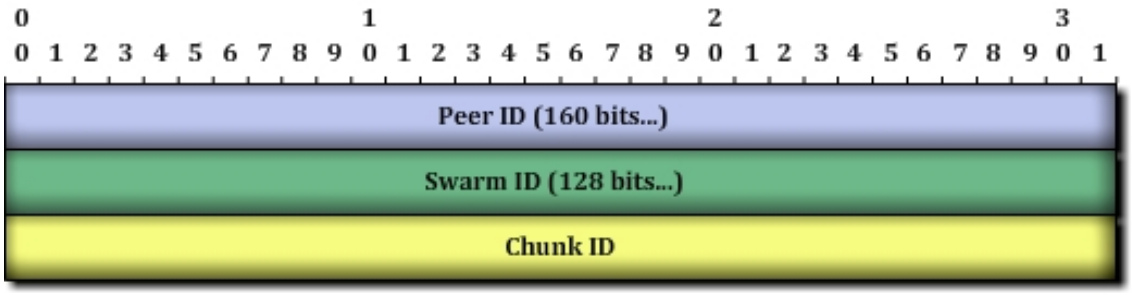

Fig. 4.11.- Cos del missatge FIND de petició

Els missatges de tipus FIND, a diferència de la majoria de missatges de petició, si que tenen un cos de missatge a la resposta. A dins de l'apartat de "Missatges de resposta" es veu detalladament com ha de ser aquest cos de missatge.

4.3.7. KEEPALIVE Message

Els missatges KEEPALIVE són missatges enviats periòdicament des dels *peers* cap al *Tracker* per notificar a aquest de que el *peer* encara està viu. Si un *Tracker* no rep un KEEPALIVE dins d'un temps determinat pre-configurat, el *Tracker* assumirà que el *peer* ja no està disponible i realitzarà les mateixes operacions lògiques que en un LEAVE, és a dir, l'esborrarà de la llista de *peers* de l'eixam que correspongui.

Per a formar un missatge de tipus KEEPALIVE, el *peer* construeix la capçalera comú de missatge i l'omple amb els valors corresponents. El camp *Method* es posarà a KEEPALIVE, que correspon a la codificació 0x0806. El cos de missatge, en aquest cas, es compondrà d'un únic camp que serà el *Peer ID*.

Aquest camp de *Peer ID*, serà un camp de 160 bits que contindrà l'identificador del *peer* que fa la petició de KEEPALIVE, de forma que el *Tracker* al rebre el missatge tindrà l'identificador d'aquest *peer*.

La resposta que donarà el *Tracker* no inclourà cap tipus d'informació addicional, sinó que només servirà per indicar al *peer* que ha rebut el missatge i si hi ha hagut algun problema o no.

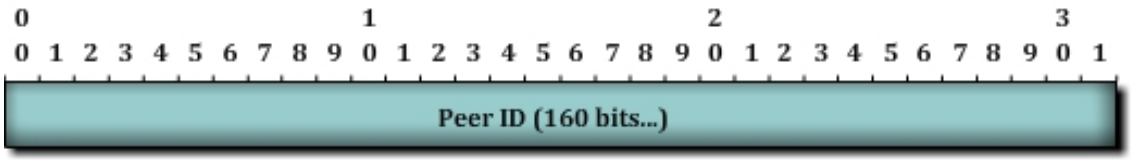

Fig. 4.12.- Cos del missatge KEEPALIVE

4.3.8. STAT Messages

Hi ha dos tipus d'STAT *Messages*, STAT_QUERY i STAT_REPORT. Els dos són mètodes que funciones en dues direccions, de *peer* a *Tracker* i de *Tracker* a *peer*, però es diferencien en que el primer d'ells s'usa per demanar informació estadística i el segon per reportar informació.

Abans d'entrar en detall a descriure ambdós missatges, es mostra una taula amb els *Property Types* per missatges STAT. Les següents estadístiques es mostren com a exemples d'informació que poden ser útils, en els que s'han definit uns valors de *Property Types*. A mesura que avanci el desenvolupament del protocol, aquests valors poden veure's modificats. Si el *Property Value* està buit, el valor és un booleà, la presencia del qual indica que és cert.

P-Type	Definicions/Descripció
0x0000	Caching size: mida disponible per caching
0x0001	Bandwidth: ample de banda disponible
0x0002	Link numbers: enllaços acceptables per un peer remot
0x0003	Certificate: certificat del peer
0x0004	NAT/Firewall: peer creu que està darrere NAT (El camp Property Value està buit)
0x0005	STUN: El peer suporta STUN Service (El camp Property Value està buit)
0x0006	TURN: El peer suporta TURN Service (El camp Property Value està buit)
0x0007	Sum Volume: Suma de bytes de dades rebudes dels peers del sistema d'streaming
0x0008	Acces Mode: ADSL/Fibra/GPRS/3G/LTE/WiFi/etc.
0x0009	End Device: STB/PC/Telèfon Mòbil
0x000A	Avaliable Battery Level

Fig. 4.13.- Property Types per missatges STAT

4.3.8.1. STAT_QUERY Message

Com ja s'ha comentat, aquest mètode funciona en dues direccions. De *peer* a *Tracker* permet als *peers* consultar informació estadística sobre l'estat del sistema, i potser sobre *peers* particulars. De *Tracker* a *peer*, en canvi, permet al *Tracker* demanar informació estadística d'un *peer* en particular. És a dir, aquest mètode permet als *Trackers* recopilar dades estadístiques per a millorar el rendiment del sistema. STAT_QUERY és un mètode opcional.

Per a formar un STAT_QUERY *message*, el *Tracker* construeix la capçalera general de missatge. El remitent omple adequadament els diferents camps de la capçalera, posant el camp *Method* a STAT_QUERY, que correspon al valor codificat 0x0807. La resta de

paràmetres de la capçalera s'omplen com ja s'indica al seu apartat corresponent. El cos de missatge d'un STAT_QUERY té els següents paràmetres:

- **Nombre de P-Types:** O *Parameter Types*. Camp de 16 bits que codifica el nombre de *P-Types* que hi haurà a continuació, no la longitud d'ells. El nombre de *P-Types* pot ser major o igual a zero. Ja que el missatge és petició, això no són TLVs, sinó una llista de *P-Types* de la que el *Tracker* està demanant informació.

- **Reserved:** Camp de 16 bits reservat per a futurs usos.

- **P-Type i P-Type Addicionals:** Camp/s de 16 bits que indica el valor del *P-Type* corresponent. El nombre de camps d'aquest tipus ve determinat pel valor del camp "Nombre de *P-Types*" i pot ser més gran o igual que zero. Aquests zero o més *P-Types* que el *Tracker* vol que el *peer* reporti, són llistats. Són peticions, no TLVs, així que simplement són una llista de *P-Types* de la que el *Tracker* està demanant informació.

- **Reserved (per senars):** Camp de 16 bits que només s'utilitzarà si el nombre de *P-Types* és senar, ja que la mida del cos de missatge no seria un múltiple de 32. És un camp de farciment que no té cap altra utilitat a part d'aquesta.

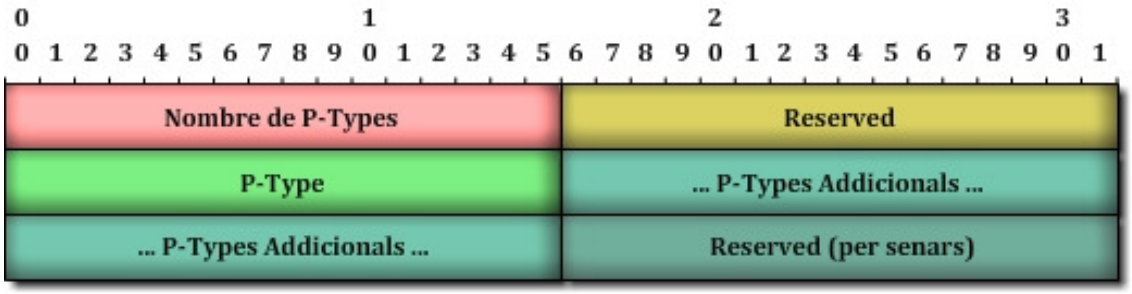

Fig. 4.14.- Cos de missatge d'STAT_QUERY

A l'hora de respondre, amb els missatges STAT_QUERY passa el mateix que amb els de tipus FIND, i es que a diferència de la majoria de missatges de petició, si que tenen un cos de missatge a la resposta. A dins de l'apartat de "Missatges de resposta" es veu detalladament com ha de ser aquest cos de missatge.

4.3.8.2. STAT_ REPORT Message

El mètode STAT_REPORT, un dels dos tipus de missatge STAT, funciona també en dues direccions. De *peer* cap a *Tracker* permet als *peers* presentar dades estadístiques a la millora de rendiment del sistema del *Tracker*. De *Tracker* a *peer* permet notificar la

informació general de tot el sistema PPSP, com poden ser els *chunks* més estranys, l'ample de banda de descarrega mig, etc.

Per a formar un STAT_REPORT *message*, el *Tracker* construeix la capçalera general de missatge. El remitent omple adequadament els diferents camps de la capçalera, posant el camp *Method* a STAT_ REPORT, que correspon al valor codificat 0x0808. La resta de paràmetres de la capçalera s'omplen com ja s'indica al seu apartat corresponent. El cos de missatge està format pels següents paràmetres:

- **Peer ID:** Camp de 160 bits que indica l'identificador del *peer* que fa la petició de missatge. Aquest identificador l'ha aconseguit prèviament el *peer* amb qualsevol mecanisme *out-of-band*.

- **Nombre d'Stats:** Camp de 16 bits que indica el nombre d'estructures STAT TLV que hi haurà a continuació del camp *Reserved*. No es refereix a la seva longitud. El nombre d'*Stats* pot ser major o igual a zero.

- **Reserved:** Camp de 16 bits reservat per a futurs usos.

- **STAT TLVs i STAT TLVs Addicionals:** Camp/s que contenen les estructures de tipus STAT TLV. El nombre d'STAT TLVs és major o igual a zero i ve determinat pel camp "Nombre d'*Stats*". L'estructura d'un STAT TLV és defineix seguidament de la figura que representa el cos de missatge.

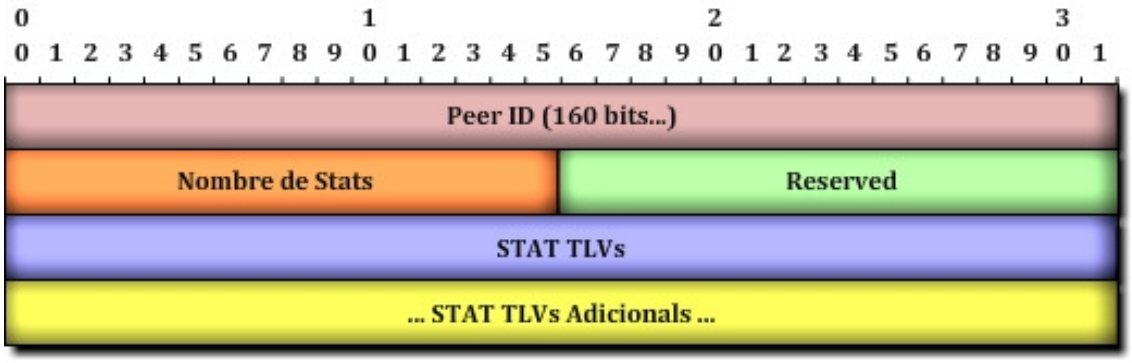

Fig. 4.15.- Cos de missatge d'STAT_REPORT

L'estructura dels STAT TLV és de la forma següent:

- **P-Type:** Camp de 16 bits que indica el valor de *P-Type*. Els valors que pot prendre el camp s'indiquen anteriorment, a la taula de valors de *P-Type*.

- **Longitud:** Camp de 16 bits que indica la longitud del camp *Property Value* en bits, limitant els valors a 8000.

- **Property Value:** Camp que pot variar des de 0 bits, si el camp *Value* de *P-TYPE* està definit com buit i per tant la longitud es fixarà a zero, fins a 8000 bits, que és la longitud màxima que pot tenir. El *peer* crea una llista de propietats que vol compartir amb el *Tracker*, ho ensambla en un missatge i transmet el missatge cap al *Tracker*.

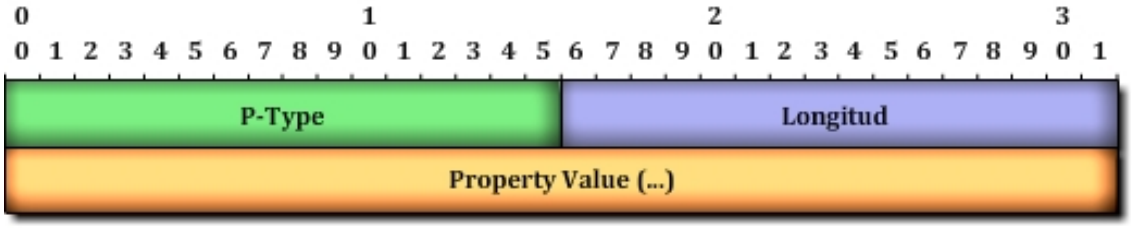

Fig. 4.16.- Format de TLV del peer property

La resposta que donarà el *Tracker* a un missatge de tipus STAT_REPORT no inclourà cap mena d'informació addicional, sinó que només servirà per indicar al *peer* que ha rebut el missatge i si hi ha hagut algun problema o no.

4.4. Missatges de resposta

Els missatges de resposta són l'altre tipus de missatge que conforma el cos (*Message Body*) dels missatges que s'envien amb el *PPSP Tracker Protocol*. Són missatges que es generen per respondre a un missatge de petició, i que seran en funció d'aquesta petició, ja sigui correcta, incorrecta o amb qualsevol altra característica, com pot ser la sintaxis invàlida o una versió incorrecta del protocol, per posar un parell d'exemples.

És a dir, per correspondre amb una petició feta, hi ha un cert nombre de respostes lògiques a aquests missatges per retornar. En cas de que es parli de la codificació binaria, això inclou missatges *malformed* i peticions. Si es parla de la proposta de codificació mitjançant HTTP/XML, el propi protocol HTML s'encarregarà dels missatges mal formats, però cossos de missatges amb format incorrecte per XML generaran segurament errors de nivell al *Tracker Protocol*. Aquests continguts seran reportats a un missatge HTTP.

En general, les respostes poden contenir llistes de *peers*, informació estadística i metadades. Actualment, no hi ha definit un format específic per aquest tipus

d'informació de forma oficial. Partirem doncs, d'una possible implementació de com podrien ser aquests missatges.

Igual que passava amb els missatges de petició, els de resposta estaran formats per una capçalera, que serà general per tots i igual que la dels missatges de petició, amb els corresponents camps, i un cos de missatge. Per identificar quin tipus de missatge, a la capçalera hi ha el camp *Method*, que ens indicarà de quin tipus de missatge es tracta. La diferent informació que es retornarà amb els diferents missatges, serà el que conformarà el cos del missatge.

A continuació, es fa una descripció dels diferents tipus de missatge de resposta amb els que ens trobem.

- **SUCCESFULL (OK)**: És un missatge de resposta que indica que el missatge de petició s'ha processat correctament i l'operació corresponent ha sigut completada. En el cas de que el missatge sigui una petició d'informació, el cos del missatge també inclourà l'informació desitjada per retornar. Així doncs, la següent informació serà retornada per cada tipus de missatge de petició:

    - Pels missatges de petició de tipus CONNECT, DISCONNECT, JOIN, JOIN_CHUNK, KEEPALIVE i STAT_REPORT, es retornarà qualsevol informació d'estat requerida sobre l'operació feta amb èxit.

    - Si el missatge és del tipus FIND, es retornarà la llista dels *peers* que acompleixen els criteris de recerca desitjats. A més, metadades sobre l'eixam es pot retornar amb la resposta igualment.

    - Al missatge de tipus STAT_QUERY es respon amb les estadístiques demanades al cos del missatge.

  Als apartats 5.4.1, 5.4.2 i 5.4.3 es detallaran com seran aquestes respostes, ja que els missatges SUCCESFUL són els únics que retornaran com a resposta informació addicional (no en tots els cassos) i no només per indicar que hi ha hagut un problema, com són la resta de missatges de resposta.

- **INVALID SYNTAX:** Missatge de resposta que serveix per indicar que hi ha un error de format al missatge de petició.

- **VERSION NOT SUPPORTED:** Tal i com diu el nom, és un missatge de resposta que es generarà quan la versió del *Tracker* Protocol del missatge de petició no sigui correcta. Actualment, la única versió del protocol que es suporta es la 0.1, així que qualsevol altra versió serà considerada *not supported*.

- **MESSAGE NOT SUPPORTED:** Quan una petició en particular no es suporta pel *Tracker*, es genera com a resposta un missatge d'aquest tipus. Com a exemple, alguns *Trackers* poden escollir no implementar les funcions estadístiques.

- **TEMPORARILY OVERLOADED:** Missatge de resposta per indicar que el servidor està temporalment sobrecarregat i no pot processar el missatge de petició rebut en aquest moment.

- **INTERNAL ERROR:** Missatge de resposta per indicar que degut a un error intern, el servidor no pot processar el missatge de petició rebut en aquest moment.

- **MESSAGE FORBIDDEN:** Missatge per indicar que la petició feta no està permesa. Algunes de les peticions fetes pels *peers*, com per exemple consultar informació estadística del *Tracker*, no estan permeses.

- **OBJECT NOT FOUND:** Missatge de resposta per indicar que l'objecte demanat no es pot trobar. Aquest objecte pot ser un eixam al que el *peer* es vol unir o que simplement ha buscat per demanar algun tipus d'informació.

- **AUTHENTICATION REQUIRED:** Poden existir serveis que requereixin autenticació per accedir a l'informació. Aquest missatge de resposta és el que indica.

- **PAYMENT REQUIRED:** Hi poden existir serveis exclusius que requereixin pagament per tal de poder accedir-hi. Si es dóna el cas, aquest missatge serà la resposta.

Cal afegir que els missatges de tipus TEMPORARILY OVERLOADED, INTERNAL ERROR, MESSAGE FORBIDDEN, OBJECT NOT FOUND, AUTHENTICATION REQUIRED i PAYMENT REQUIRED, seran tractats al nivell HTTP en el cas de que el tipus de la codificació sigui HTTP/XML i no binària.

4.4.1. Respostes sense informació addicional

Els missatges de petició de tipus CONNECT, DISCONNECT, JOIN, JOIN_CHUNK, KEEPALIVE i STAT_REPORT, independentment de que hagin tingut èxit o no, un cop es fa la resposta des del *Tracker*, no han de portar informació addicional. És a dir, el missatge de resposta es compondrà de la capçalera general de missatge i no inclourà cos de missatge. Per tant, aquestes respostes seran totes de 192 bits i la capçalera del missatge de resposta serà la mateixa que la del missatge de petició, amb l'excepció de que el camp *Method* serà diferent i en comptes d'indicar el tipus de missatge de petició que s'ha fet, indicarà de quin tipus de missatge de resposta es tracta.

4.4.2. Respostes a missatges de tipus FIND

Com ja s'ha comentat, els missatges FIND són els que usen els *peers* per demanar al *Tracker* que els hi retorni llistes de *peers* que tenen un contingut específic o que estan a un eixam concret. El cos de missatge de la resposta es composa dels següents camps:

- **Swarm ID:** Camp de 128 bits que portarà el valor de l'eixam al qual pertanyen els *chunks*.
- **Nombre de *peers*:** Camp de 16 bits per indicar, no la longitud de la llista, sinó el nombre de *peers* que apareixeran llistats a la llista de *peers*.
- **Reserved:** Camp de 16 bits reservat per a futurs usos.
- **Llista de peers:** Camp de valor múltiple de 32, encara en discussió. El debat està en si el camp hauria d'incloure la llista amb les adreces IP o els identificadors de *peer*. Amb el *Peer ID*, el *peer* que fa la petició pot trobar i establir connexió amb els *peers* candidats enrutant a través d'*overlay*, usant un mecanisme com RELOAD. Per ara, considerarem que el camp estarà format pels diferents *Peer ID*.

Fig. 4.17.- Cos del missatge FIND de resposta

4.4.3. Respostes a missatges de tipus STAT_QUERY

STAT_QUERY s'usa per demanar informació estadística. De *peer* a *Tracker* permet als *peers* consultar informació estadística sobre l'estat del sistema. De *Tracker* a *peer* permet al *Tracker* demanar informació estadística d'un *peer* en particular. Permet als *Trackers* recopilar dades estadístiques per a millorar el rendiment del sistema.

Quan un missatge STAT_QUERY es rebut pel *peer*, aquest ha de respondre amb una de les codificacions de resposta conegudes, que serà el mètode per posar al camp *Method* de la capçalera.

El cos de missatge conté un TLV per cada estadística demanada. Els paràmetres del cos de missatge de resposta són:

- **Peer ID:** Camp de 160 bits que indica l'identificador del *peer* que fa la petició de missatge. Aquest identificador l'ha aconseguit prèviament el *peer* amb qualsevol mecanisme *out-of-band*.
- **Nombre d'Stats:** Camp de 16 bits que indica el nombre d'estructures STAT TLV que hi haurà després del camp *Reserved*, no la longitud d'aquestes. El nombre d'estructures pot ser més gran o igual que zero.
- **Reserved:** Camp de 16 bits que es reserva per usos futurs.
- **STAT TLVs i STAT TLVs addicionals:** Camps de 32 bits que consistiran, cadascun dels que hi hagi, en una estructura STAT TLV. El nombre de camps que hi haurà vindrà determinat pel camp "Nombre d'*Stats*", que pot ser un nombre major o igual a zero. L'estructura dels STAT TLVs és la mateixa que hem vist al missatge de petició STAT_REPORT.

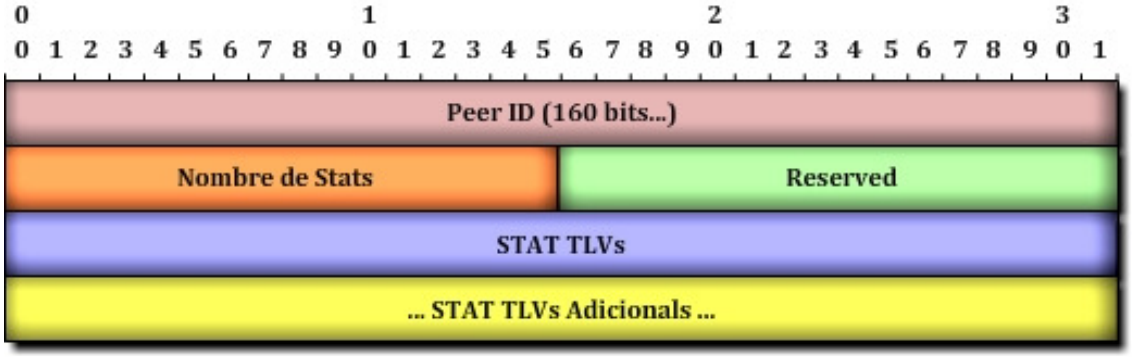

Fig. 4.18.- Cos de missatge d'STAT_QUERY de resposta

5. PPSP Peer Protocol

5.1. Introducció

Havent vist amb detall el *PPSP Tracker Protocol*, es pot passar a parlar de l'altre protocol important que composa PPSP. *PPSP Peer Protocol* s'encarrega d'estandarditzar el format, així com la codificació i els missatges de tota la informació que circula entre els diferents *peers*.

Aquesta informació inclourà, com a mínim, un mapa de bits indicant de quins *chunks* disposa un *peer*, els identificadors de *chunk* requerits, la preferència del *peer* indicant quin tipus de *peers* candidats prefereix el *peer* que fa la petició, negociació de protocol de transport i, finalment, alguna informació que pugui servir com ajuda per a millorar l'actuació de PPSP. Una de les opcions per a realitzar el *Peer Protocol* és reutilitzar part del protocol RELOAD.

Cal tenir en compte, que al igual que el *PPSP Tracker Protocol*, aquest es un protocol que encara està en discussió, tot i que a diferència de l'anterior, el *PPSP Peer Protocol* just acaba de veure la seva primera versió de la definició. Això implica que el protocol està en un estat molt més prematur, on moltes de les parts del mateix estan encara sense acabar, per discutir o ni s'han tingut presents.

El que hi ha a continuació és, doncs, una primera aproximació del que haurà de ser aquest protocol, partint de la definició feta al *draft* corresponent, així com d'algunes parts agafades de la definició feta del *PPSP Tracker Protocol*, que està en un estat més avançat de desenvolupament, i que presumiblement poden compartir ambdós protocols.

5.2. Arquitectura i entitats funcionals

Un cop obtinguda la llista de *peers*, ja sigui des de *Trackers*, *peers* remots o qualsevol altre mitjà, el *peer* que l'ha rebut ja podrà comunicar-se amb els *peers* remots llistats. L'únic participant del *PPSP Peer Protocol* és el *peer*. L'informació intercanviada entre els *peers* s'utilitza també per ajudar als *peers* a obtenir el contingut desitjat i es veurà definida durant el subapartat que hi ha a continuació d'aquest, als requeriments. Aquesta arquitectura mostra tots els possibles mètodes de comunicació i els diferents tipus d'informació que hi poden haver entre els *peers*, tot i que això no implica que tots ells estiguin dintre de l'àmbit del protocol.

A la figura següent, es pot veure un esquema de comunicació entre dos *peers*, implementant ambdós el *PPSP Tracker Protocol*, vist anteriorment, i el *PPSP Peer Protocol*, que és el que ens ocupa ara; així com veient esquemàticament els fluxos de dades de senyalització i de localització que s'intercanvien amb el *PPSP Peer Protocol*.

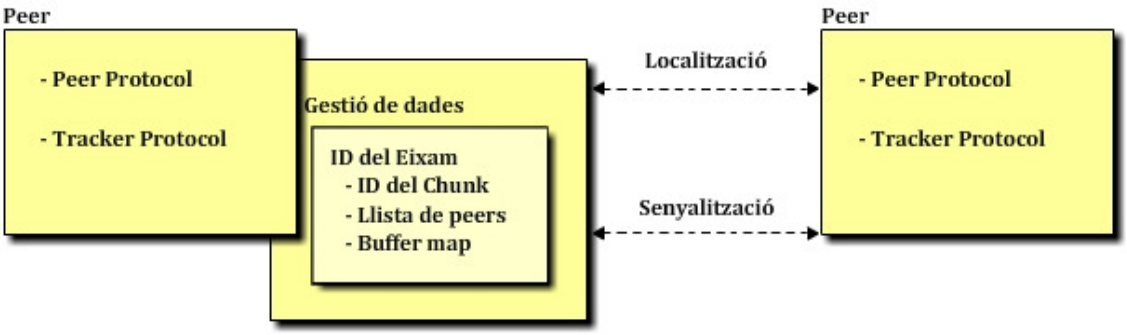

Fig. 5.1.- Entitats funcionals del PPSP Peer Protocol

Descrivint com seria el flux general d'un missatge del *Peer Protocol*, un *peer* obté la llista de *peers* a través del *Tracker*, d'un *peer* o qualsevol altre mètode, i llavors tria alguns *peers* candidats d'aquesta llista per fer la petició de *peers* addicionals. Els *peers* intercanvien un tipus de dades anomenat *Data Availability* amb alguns *peers* remots amb la intenció de trobar quin d'aquests *peers* tenen el contingut desitjat pel *peer* en qüestió. La forma de triar un *peer* remot està fora de l'abast del *Peer Protocol*.

D'acord amb les dades del *Data Availability*, el *peer* escull els *peers* remots que contenen el contingut desitjat i envia unes peticions de descarrega (*Download Requests*) cap als *peers* remots. Si la resposta que s'obté per part dels *peers* remots és positiva, fet que implica que aquests *peers* estan disposats a proporcionar el contingut triat, el *peer* intentarà establir una connexió de transport amb els *peers* remots per tal de que les dades puguin ser transmeses. Finalment, un cop s'han descarregat les dades, la connexió es tancarà. Aquest procés es repetirà fins que el *peer* obtingui tot el contingut que desitgi.

PPSP Peer Protocol es divideix en dues parts ben diferenciades, cadascuna d'elles formada per un protocol independent. La primera d'aquestes dues parts, correspon al protocol que s'encarrega de la localització dels *peers*, anomenat *Peer Locating Protocol*. La segona de les parts del *Peer Protocol* és la que correspon al protocol senyalització, el *Peer Signaling Protocol*.

Peer Locating Protocol ens implica que el *peer* que fa la petició necessita un mecanisme per connectar amb els *peers* remots de la llista de *peers*. Això vol dir que aquesta part en concret està força lligada amb el *PPSP Tracker Protocol*. En quant als

diferents *peers* identificats a la llista de *peers*, el mecanisme de localització i el corresponent protocol seran diferents. Si els identificadors de *peers* estan inclosos a la llista de *peers*, el *Peer Locating Protocol* utilitzarà el protocol RELOAD o dissenyarà un nou *Peer* to *Peer Locating Protocol*. Si en canvi, s'utilitzen URIs, *Peer Locating Protocol* utilitzarà el protocol SIP o un mecanisme similar per a localitzar i establir sessions amb altres *peers*.

Amb *Peer Signaling Protocol*, en canvi, els *peers* poden intercanviar informació tal com mapes de bits i llisters de *peers* addicionals, la possibilitat de negociar els protocols de transport que s'usaran i, també, preguntar per *chunks* o eixams desitjats.

Ambdues parts del *PPSP Peer Protocol* són components importants i si estan definides de forma independent és amb el propòsit de assegurar que les dues parts es poden desenvolupar de forma sincronitzada i eficient.

5.3. Requeriments

En aquest subapartat es veuran diversos requeriments que seran necessaris per a la elaboració del protocol. No importa quina opció, o opcions, de combinacions es faci servir com a decisió final, aquests són uns requeriments generals necessaris que hauran de ser complits pel *Peer Protocol*.

5.3.1. Localització i connexió

Tal i com s'ha comentat prèviament, la localització amb *Peer Locating Protocol* està fortament lligada amb el *PPSP Tracker Protocol*, vist en l'apartat anterior. Un *peer* obtindrà la llista de *peers* mitjançant el *Tracker*, amb els seus identificadors de *peer*, adreces IP o qualsevol altre identificador de *peer*, la forma dels quals influirà el mecanisme de localització. Tal i com hem definit el *PPSP Tracker Protocol*, un cop es retorna la llista de *peers*, la informació rebuda d'aquests *peers* va lligada amb l'identificador de *peer* i no amb l'adreça IP.

La idea és que el *peer* sigui capaç de localitzar els *peers* de la llista de *peers* i connectar amb ells amb la menor ajuda, o sense aquesta, del *Tracker*. Per tant, *Peer Protocol* ha de proporcionar un mecanisme per connectar-se entre ells.

5.3.2. Intercanvi d'informació

Després d'una connexió realitzada amb èxit entre *peers*, aquests intercanviaran informació que els ajudarà a decidir d'on han de descarregar el contingut desitjat.

5.3.2.1. Peerlist Sharing

En molts protocols d'*streaming peer* to *peer*, com *PPLive* o *PPStream*, el client pot obtenir la llista de *peers* tant del *Tracker* com dels *peers* remots. La funció principal del *Tracker* és la de recopilar tots els *peers* en un eixam en particular. Tot i això, a la pràctica, els *Trackers* coneixen tots els *peers* candidats que hi ha en un eixam en particular, especialment quan hi ha massa *peers* en un mateix eixam. En molts casos, el *peer* no vol rebre tots els *peers* candidats, només una part. En alguns casos, el *Tracker* només proporciona una llista de *peers* de referència a través dels quals, un *peer* que es connecti, pot rebre més llistes de *peers* si ho desitja. Tot aquest sistema de compartició és el que es coneix com a *Peerlist Sharing*, i entre les seves característiques es poden trobar:

- **Peerlist Request:** *PPSP Peer Protocol* ha de permetre que un *peer* pugui enviar peticions per a obtenir la llista de *peer* dels seus *peers* remots sobre un contingut en particular, sigui uns *chunks* o un eixam. Aquesta característica, tot i que el *peer* no demani la llista a altres *peers* (sinó al *Tracker*, amb el *PPSP Tracker Protocol, o usant altres mètodes*), ha d'estar garantida al *peer*.

- **Peerlist Response:** *PPSP Peer Protocol* ha de permetre poder crear una resposta a la petició de la llista, portant aquesta llista de *peers*. Al rebre una petició del tipus *Peerlist Request*, un *peer* remot escollirà un seguit de candidats de la seva llista de *peers* local i els encapsularà en la *Peerlist Response*, fent una nova llista de *peers* que anirà a parar al *peer* que havia fet la petició prèviament.

- **Peerlist Parameter:** *PPSP Peer Protocol* ha de permetre als *peers* indicar el nombre de *peers* que hi ha a una llista de *peers* quan demanin una llista de *peers* a altres *peers*. A més, també ha de permetre als *peers* indicar les seves pròpies característiques quan un *peer* remot sigui el que li faci una petició demanant una llista de *peers*.

5.3.2.2. Data Availability

PPSP Peer Protocol ha de permetre als *peers* fer peticions per demanar la disponibilitat de dades, és a dir, per la *Data Availability* dels *peers* remots. A més, el protocol ha de deixar que els *peers* puguin portar ells mateixos aquesta *Data Availability* a les respostes fetes a una petició de *Data Availability*. És a dir, el protocol farà que un *peer* pugui fer una petició de *Data Availability* a un *peer* remot, i aquest *peer* remot serà

capaç de portar aquest tipus de dades a la resposta, que pot portar la *Data Availability* de formes diverses al cos del missatge.

A més de tot això, el protocol ha de permetre portar diferent estructures de dades per usar a diferents tipus d'aplicacions. Això es déu a que el model de dades de cada aplicació pot ser diferent. La llista de *peers* intercanviada entre *peers* també pot ser diferent.

5.3.2.3. Property Exchange

PPSP Peer Protocol ha de permetre als *peers* intercanviar les propietats dels *peers*. Aquest requeriment permet al *peer* indicar les seves propietats quan es fa una petició de la llista de *peers*. Amb aquesta funcionalitat, un *peer* pot intercanviar propietats amb el *peer* candidat obtingut, ja sigui a través del *Tracker* o mitjançant els *peers* remots.

5.3.3. Estat de la connexió

El protocol, partint de requeriments influenciats per les especificacions de *BitTorrent*, desenvolupa un protocol que s'encarregarà del control de la congestió: *Application-level Congestion Control*. *PPSP Peer Protocol* ha de permetre a un *peer* notificar al seu *peer* remot si vol fer un *upload* de contingut cap al *peer* remot.

Això és especialment significatiu quan un *peer* està sobrecarregat. Aquest *peer* sobrecarregat, que està a la llista de *peers* de molts altres *peers*, distribuït a la llista de *peers* de la resposta a la petició de múltiples *peers*, estarà ocupat fent un *upload* cap a diversos *peers* remots quan un nou *peer* connectarà amb ell i farà una petició per demanar més accions d'aquest *peer*. Si el *peer* remot no reporta cap *feedback*, el *peer* que fa la petició preguntarà una i altra vegada basant-se a la configuració de l'aplicació.

5.3.4. Funcions a temps real

El protocol ha de suportar *download* i *upload* a la vegada, és a dir, que un *peer* pot baixar dades de varis *peers* i, al mateix temps, pujar informació a un grup de *peers*.

A més, ha de tenir algun mecanisme que elimini, o al menys decrementi, tant el *jitter* com la pèrdua de paquets. Ja s'ha comentat que les aplicacions *streaming* són molt sensibles al *jitter* i a la pèrdua de paquets, fets que degraden la qualitat de l'experiència enormement, per tant s'han de tenir presents aquests fets des del protocol i evitar-los.

5.3.5. Negociació de transport

PPSP Peer Protocol serà l'encarregat de negociar entre dos *peers* el protocol de transport que faran servir per transmetre les dades entre ells, que haurà de ser compatible per ambdós *peers*. Les opcions que hi han per transmetre contingut d'*streaming* són múltiples. RTP, RSTP, FTP, UDP, TCP, MSRP, etc; així que els *peers* primer intentaran negociar un protocol de transport i un cop decidit començaria l'intercanvi de missatges de senyalització per a configurar la connexió de transport entre ells.

5.3.6. Seguretat

El protocol ha de garantir la privacitat dels *peers*, així com identificar i verificar l'identitat d'aquests. És a dir, *PPSP Peer Protocol* ha de garantir que un cop un *peer* fa una petició, aquesta només sigui rebuda pels *peers* remots als que va destinada. Això significa que el protocol ha d'evitar que el missatge sigui interceptat i/o canviat mitjançant un *man-in-the-middle*. També es pot donar el cas de que atacants es facin passar per algú altre i demanin als *peers* que enviïn dades a *peers* innocents. Aquesta és una forma comú d'atac a xarxes P2P i *PPSP Peer Protocol* ha de proporcionar mecanismes per evitar-los.

5.4. Peer Locating Protocol

Peer Locating Protocol és un dels dos protocols principals que conformen el protocol *PPSP Peer Protocol*. La seva funcionalitat és encarregar-se de la localització dels diferents *peers* de la xarxa, el que implica que el *peer* que fa la petició necessita un mecanisme per connectar amb els *peers* remots de la llista de *peers*. El que significa això és que aquest protocol està força lligat amb el *PPSP Tracker Protocol*, necessari, entre d'altres coses, per conèixer l'ubicació de la resta de *peers*.

Tal i com s'ha definit el *PPSP Tracker Protocol*, a la llista de *peers* s'inclouen els identificadors de cadascun dels *peers* i no les seves adreces IP. Aquest era un dels dilemes a l'hora de fer el disseny del protocol. En aquest cas, obtenint només els identificadors dels *peers*, el *peer* que fa la petició necessita una forma de localitzar els *peers* amb aquests identificadors presents a la llista. Per solucionar aquest inconvenient, es fa servir el protocol *RELOAD*.

RELOAD introdueix un mètode per localitzar i establir les connexions amb els *peers* remots mitjançant els identificadors dels *peers*, també coneguts com *Peer ID*, continguts a la llista de *peers*.

5.5. Peer Signaling Protocol

La segona de les parts del *Peer Protocol* és la que correspon al protocol senyalització, el *Peer Signaling Protocol*.

Tal i com succeeix amb el *PPSP Tracker Protocol*, alhora de definir un nou protocol hi ha dues formes de fer la seva codificació: binària o ASCII. La majoria de les aplicacions P2P utilitzen la codificació binària, que en aquest cas, inclourà varis tipus de missatge amb la característica comú de que tots compartiran una mateixa capçalera general de missatge. A partir d'aquesta, segons el tipus de missatge amb el que ens trobem, els missatges seran d'intercanvi d'informació, de negociació de transport o de petició de dades.

Aquesta capçalera general, tot i no estar directament definida al *draft* del protocol, es una proposta aproximada basada en com és la capçalera general de missatge del *PPSP Tracker Protocol*. De fet, s'ha optat per agafar una capçalera igual, tenint en compte com s'hauran d'omplir els diferents camps d'aquesta.

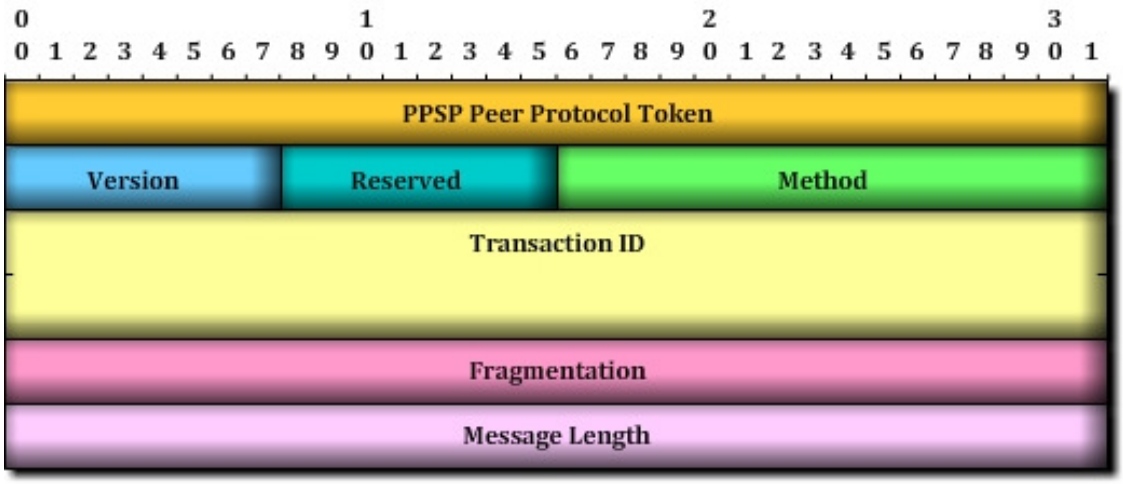

Fig. 5.2.- Proposta de capçalera general de missatge de PPSP Peer Protocol

Com es pot observar, aquesta proposta de capçalera correspon a la capçalera del *Tracker Protocol*. Aquesta ve formada per diversos camps que seran informació comú a tots els missatges. Aquests camps que conformen la capçalera són:

- **PPSP Peer Protocol Token**: De la mateixa forma que passava amb el *Tracker Protocol*, és un camp fix de 32 bits que indica al receptor que aquest es un missatge corresponent a PPSP. El valor que ha de tenir sempre aquest camp al *Tracker Protocol* es correspon a 0xd0505374, o el que és el mateix, la cadena "PPSt" amb el bit d'ordre més alt del primer byte activat. En cas de

que tant *Peer Protocol* com *Tracker Protocol* utilitzin la mateixa codificació, per tal de fer-ho més ampli es podria posar el valor més genèric, com per exemple "PPSP".

- **Version**: Camp de 8 bits que indicarà la versió del protocol que s'està usant. Per motius de compatibilitat, la versió serà un valor fix ubicat entre 0.1 i 25.4, mateixos valors que es poden trobar al camp *Version* de *PPSP Tracker Protocol*; tot i que a la versió a la que es troba el protocol actualment, el camp *Version* ha de ser posat forçosament a 0.1.

- **Reserved**: 8 bits que estan reservats, com bé diu el seu nom, per a usos especials, com podria ser per estendre el *namespace* quan aquest estigui exhaust. Els valors del camp *Reserved* no es contemplen encara.

- **Method**: Camp de 16 bits que serveis per indicar el tipus de mètode del missatge. És el camp que indica de quin tipus serà el *Message Body* que hi haurà a continuació de la capçalera del missatge. Els diferents valors que pot adoptar el camp *Method* no s'han definit encara, però la seva codificació serà semblant a la que s'ha adoptat amb el camp *Method* del *PPSP Tracker Protocol*.

- **Transaction ID**: El conforma un camp de 64 bits que identifica la transacció i també permet als receptors que discerneixin transaccions que d'altra forma serien idèntiques. Les respostes usen el mateix *Transaction ID* que la petició a la que corresponen. A més a més, el camp *Transaction ID* també s'utilitza per a rejuntar fragments.

- **Fragmentation:** Camp format per 32 bits que indica si el missatge s'ha fragmentat en varies parts, degut a la seva longitud, o no.

- **Message Length:** Aquest camp indica la longitud del missatge en bytes, incloent la pròpia capçalera. És un camp de 32 bits. En el cas de que un missatge estigui fragmentat, el camp *Message Length* indica la mida de cadascun dels missatges per separat, concretament el que li correspongui, i no la suma total de la mida dels fragments.

En cas de que la codificació usada no sigui la binària sinó la ASCII, els tipus de missatge diferents, així com la capçalera general de missatge, es mantenen. La diferència radicarà al tipus de codificació, que serà basada en HTTP/XML.

6. P2P Streaming per nodes mòbils

6.1. Introducció

Com ja s'ha comentat, el grup de treball de PPSP desenvolupa diferents protocols per sistemes d'*streaming P2P*. Les solucions anteriors estaven destinades generalment a connexions cablejades o fixes. Tenint en compte que s'espera que les comunicacions mòbils P2P adoptin un creixement ràpid i la natura dels nodes mòbils i dels entorns mòbils ocasionen reptes específics a les comunicacions P2P, específicament a escenaris *streaming*.

Aquests escenaris on les xarxes *P2P Streaming* contenen nodes mòbils requereixen una consideració especial de les capacitats de dispositius mòbils: ample de banda d'*uplink* enfront del de *downlink*, durada de la bateria, interfícies múltiples, potència de processament, *Geo-targeting*, etc.

6.2. Problemes dels nodes mòbils

Els nodes mòbils és veuen limitats per naturalesa degut a la seva bateria limitada, la mida de la pantalla, la capacitat computacional, etc. A més, els nodes mòbils operen en entorns variables i impredictibles. Aquests diversos atributs provoquen diferents problemes que poden afectar de forma adversa les sessions d'*streaming P2P*.

6.2.1. Ample de banda d'Uplink enfront el de Downlink

En general, els nodes mòbils tenen capacitats d'ample de banda asimètric, és a dir, que la velocitat de *uplink* és diferent a la de *downlink*. Normalment, la majoria dels nodes mòbils són capaços de suportar majors taxes de bit a l'enllaç de baixada (cap al dispositiu mòbil), que a l'enllaç de pujada (des del dispositiu mòbil).

A més d'això, moltes xarxes mòbils (per exemple 2G i 3G) també tenen polítiques per assignar l'ample de banda d'aquesta forma asimètrica independentment de la capacitat del node en qüestió.

Degut a que les sessions *peer-to-peer streaming* poden ser tant generades com terminades a un node mòbil, aquesta asimetria de l'ample de banda s'ha de tenir en consideració a dins del *PPSP Tracker Protocol* (com a part dels paràmetres d'estat del *Peer* que es reporten cap al *Tracker*), i pot afectar també al *PPSP Peer Protocol* a l'hora de la negociació d'informació de *peer*.

6.2.2. Durada de la bateria

Per definició, un node mòbil està sovint desconnectat de la xarxa elèctrica i funcionant amb la seva pròpia bateria. Dins d'aquest escenari, l'usuari del dispositiu mòbil pot voler restringir els tipus de les sessions de P2P a les que el node mòbil participarà degut a problemes de descàrrega de bateria. Per posar un exemple, l'usuari pot estar disposat a participar a una sessió de P2P si l'usuari està visualitzant el contingut a l'hora, però no voler participar en compartir grans quantitats de continguts cap a altres *peers*.

Per tant, la durada de la bateria d'un node mòbil s'ha de tenir present tant al *PPSP Tracker Protocol* com al *PPSP Peer Protocol*; com a part dels paràmetres d'estat del *peer* que es reporta cap al *Tracker* i altres *peers*.

6.2.3. Interfícies múltiples

Un escenari IP simple és refereix a aquell escenari on no hi ha protocol de mobilitat de la capa IP, com poden ser *Mobile IP* o *Proxy Mobile IP*, i cada *peer* necessita obtenir una nova adreça IP a través d'un mètode estàndard com pot ser DHCP perdent anteriorment la seva adreça IP prèvia.

Com es pot observar a la figura de la pàgina següent (Figura 3.1), quan el *Peer 1* és mou d'AN1 cap a AN2, la seva adreça IP canvia de IP1 a IP2. Això tindrà conseqüències tant a la connexió de *peer* a *peer* com amb la connexió del *peer* cap al *Tracker*. D'aquesta forma, per exemple, la comunicació entre els *peers* es pot perdre (el *Peer 2* envia incorrectament *chunks* cap a IP1 tot i que el *Peer 1* ja ha canviat a la nova IP). D'altra banda, pel que fa a la connexió amb el *Tracker*, aquesta es pot veure compromesa també, per exemple si el *Tracker* corromp les llistes de *peers* incloent una adreça IP incorrecta pel *Peer 1*.

Aquests efectes es poden atenuar si es té el node mòbil avisant dels canvis constantment al *Tracker* i als *peers* corresponents amb la seva nova adreça IP. La qüestió clau és, llavors, el balanç entre la senyalització requerida per proporcionar notificació del canvi d'adreça IP i la càrrega que això causa al sistema.

Per tant, la notificació de canvi d'adreça IP d'un node mòbil ha de ser considerada tant al protocol de comunicació entre *peers* com amb el protocol de comunicació entre *Tracker* i *peer*.

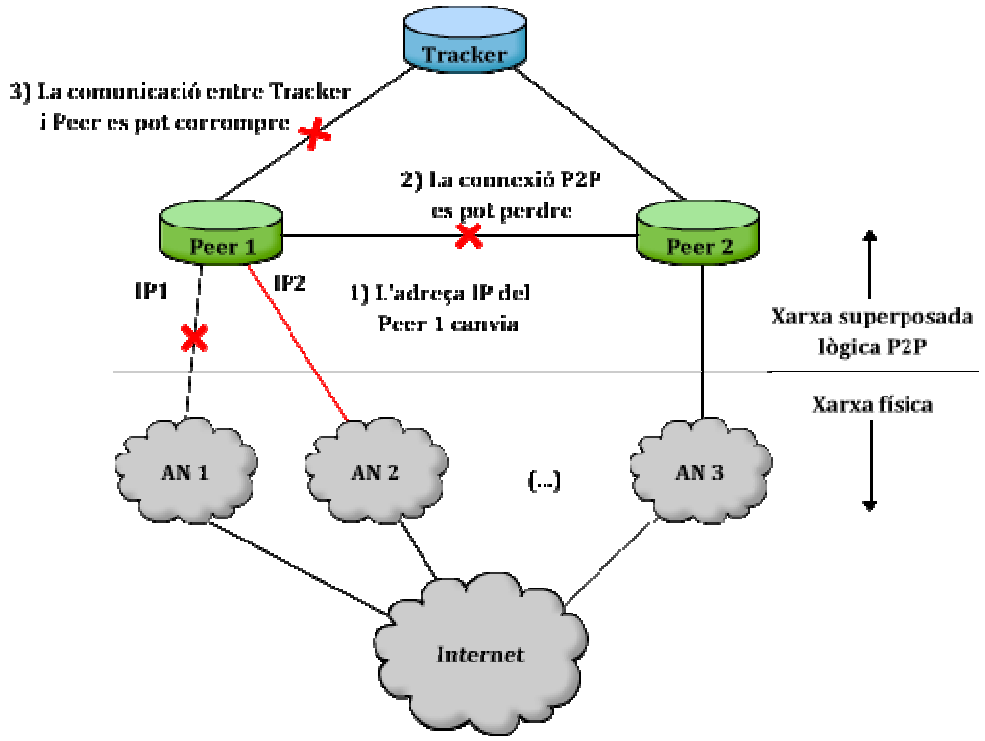

Fig. 6.1.- Streaming P2P amb dispositiu amb múltiples interfícies

6.2.4. Geo-Targeting

Geo-targeting és una tècnica que s'usa per determinar la localització física o geo-localització d'un usuari. Aquesta geo-localització o ubicació geogràfica és basa en informació de caràcter geogràfic i altre tipus d'informació personal proporcionada pel *peer* que fa la petició o una *third party*. Les tècniques per determinar l'ubicació geogràfica d'un usuari es poden recolzar en la localització cívica, coordenades geogràfiques de GPS, un identificador d'estació base cel·lular o més comunament mitjançant adreces IP. La font principal de les adreces IP amb les dades geogràfiques són els registres regionals d'Internet. Depenent de la localització, diferents regulacions i regles es poden aplicar. Per exemple, alguns continguts no podran ser distribuïts a certes ubicacions o només es podran distribuir a alguns altres llocs.

Les polítiques actuals de distribució poden aplicar certes regles per clients d'*streaming P2P* fixats. Tot i així, la mobilitat dels dispositius pot emmascarar el moviment d'una regió a una altra on és possible que hi hagi altres regles de distribució de contingut per aplicar, de forma que la representació de les polítiques establertes és d'impossible compliment. Aquest pot ser el mateix cas que si el *peer* es connecta a través d'una xarxa privada virtual (*Virtual Private Network* o VPN). En qualsevol cas, el report de la geo-localització d'un node mòbil s'haurà de considerar tant al *PPSP Tracker Protocol* com al *PPSP Peer Protocol*.

6.3. Altres consideracions mòbils

A més de les consideracions de mobilitat que s'han tingut presents a l'apartat anterior hi ha altres qüestions que s'han d'analitzar, tot i que aquestes no s'ha considerant que formin part de l'abast del protocol i de moment estan fora de la discussió per part del *PPSP Working Group*. Tot i això, es comentaran breument per tenir-ne coneixement.

6.3.1. Capacitat de processament

Alguns dispositius disposen de major capacitat que altres en termes computacionals o de potència de processament. De forma similar, els dispositius poden tenir un rendiment diferent a l'hora de generar una sessió (gravació de vídeo, per exemple) o per finalitzar-la (com la visualització de vídeo). Tenir en compte aquestes diferencies és important per mantenir una bona qualitat de la sessió d'*streaming P2P*.

6.3.2. Mobilitat de la capa d'enllaç

PPSP usa una xarxa superposada basada en P2P a la part superior de la xarxa de transport. La mobilitat o la qualitat de l'enllaç a les capes d'enllaç no són visibles de cara als *peers*.

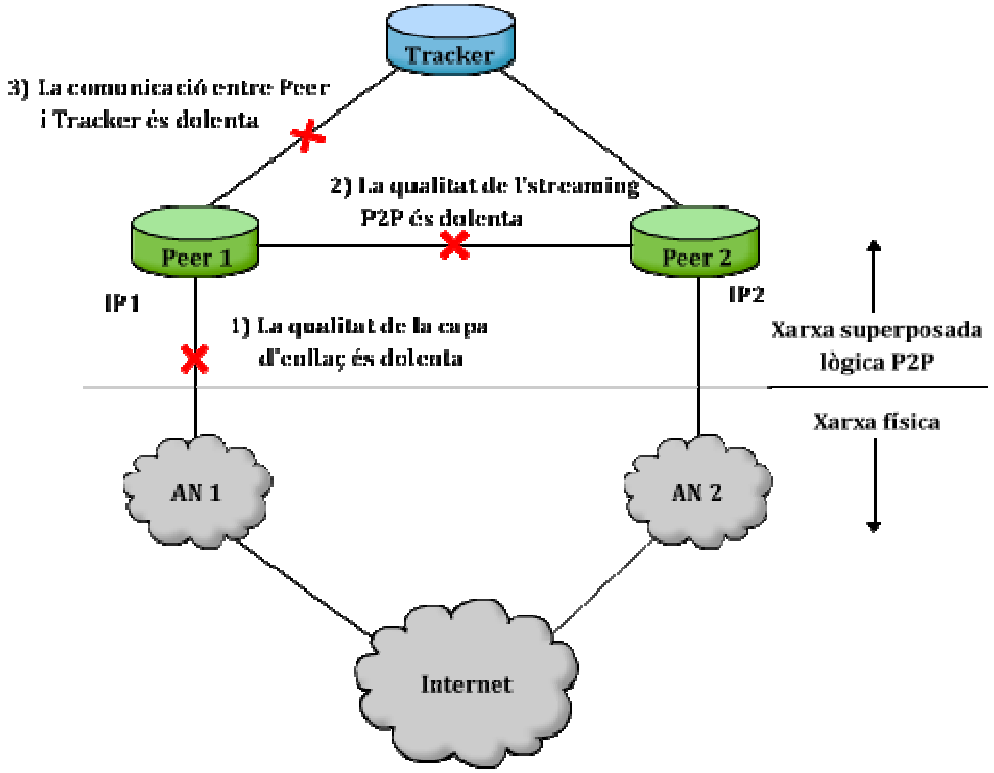

Fig. 6.2.- Streaming P2P amb mobilitat de la capa d'enllaç

Tal i com es pot apreciar a la figura anterior, si el *Peer 1* està connectat a un enllaç amb una qualitat pobre via l'accés a la xarxa mòbil 1 o *Access Network 1* (AN1), llavors el total de la qualitat de la sessió d'*streaming P2P* pot sofrir una alta taxa d'error i un rendiment baix degut a les males condicions de la capa d'enllaç. Això afectarà tant a la connexió de *peer* a *peer* com de *Tracker* a *peer*. Per exemple, la pèrdua de *frames* entre *peers*, les pèrdues de sincronisme d'àudio o vídeo o talls d'*streaming* que poden afectar als protocols de transferència del contingut mèdia.

Es veu, doncs, que si la qualitat de la capa d'enllaç que connecta el *Peer 1* amb l'AN1 (pas 1), això afecta als passos següents, de forma que la sessió d'*streaming P2P* amb el *Peer 2* també serà dolenta (pas 2), així com la comunicació d'aquest *Peer 1* amb el *Tracker* (pas 3).

6.3.3. Suport de mobilitat amb RELOAD

Ja ha quedat patent al charter proposat pel *Working Group* que qualsevol protocol desenvolupat de PPSP ha de ser compatible per interactuar amb el protocol de RELOAD. Aquest protocol de RELOAD proporciona un mecanisme de senyalització i enrutament per xarxes de superposició P2P que funcionen sobre Internet general. Caldria comentar a més, que a l'últim *draft* de RELOAD hi ha una futura secció per considerar el suport de HIP, que es un protocol experimental de mobilitat amb bones propietats de seguretat.

Com addició a HIP, els següents protocols de mobilitat també es consideraran per les interaccions entre PPSP-RELOAD: *Mobile IP* i *Proxy Mobile IP*.

6.3.4. Mobilitat del Tracker

Normalment s'assumeix que els *Trackers* són nodes fixes. Tot i això, a un entorn mòbil els nodes poden funcionar també com a *Trackers*. En aquest sentit, consideracions similars a les que s'han descrit anteriorment per *peers* mòbils es tindran presents i s'aplicaran per *Trackers* mòbils.

6.4. Protocols de mobilitat

El *Internet Protocol* (IP) és el protocol de la capa de xarxa de major èxit a l'informàtica degut als seus molts avantatges, però també té certes debilitats o inconvenients, la majoria dels quals han anat adquirint més importància segons les xarxes anaven evolucionant amb el temps. Les tecnologies com *Classless Addressing* i *Network Address Translation* s'encarreguen de combatre l'esgotament de l'espai de les adreces

IPv4, mentre que IPSec proporciona comunicacions segures de les que manca de base. Una altra debilitat de IP és que no va ser dissenyat amb dispositius mòbils en ment.

Mentre que els dispositius mòbils poden certament usar IP, la forma en la que els dispositius es tracten i s'envien els diferents datagrames enviats causa un problema quan aquests dispositius es mouen d'una xarxa a una altra. Al moment en que IP va ser desenvolupat, els ordinadors eren considerablement grans i rarament es movien. En canvi, avui en dia, tenim milions d'ordinadors portàtils i altres dispositius mòbils més petits, molts dels quals usen connexions sense cablejar per connectar amb les xarxes cablejades. Per tant, l'importància de proporcionar totes les funcions del protocol IP per tots aquests dispositius mòbils ha crescut considerablement. Per tant de suportar el protocol IP en un entorn mòbil, s'han desenvolupat un seguit de protocols, el primer d'aquests conegut com *Mobile IP Protocol*.

6.4.1. Mobile IP Protocol

Mobile IP Protocol (MIP) [14] proporciona mobilitat per IP i emmascara el moviment del dispositiu mòbil del *Correspondent Node* (CN) [15]. A aquesta secció es descriu el protocol especial desenvolupat per tal de superar els problemes dels ordinadors o dispositius mòbils vinculats a xarxes IP.

El problema bàsic amb el suport de dispositius mòbils a xarxes IP està en que l'enrutament es fa usant l'adreça IP, el que significa que l'adreça IP d'un dispositiu està lligada a la xarxa a la qual el dispositiu està localitzat. Si el dispositiu canvia de xarxa, les dades enviades a la seva adreça antiga no podran ser entregades per mitjans convencionals. Solucions tradicionals tals com l'enrutament de tota l'adreça IP o el canvi d'aquesta adreça IP de forma manual sovint creen més problemes.

Mobile IP soluciona els problemes associats amb els dispositius que canvien les localitzacions de xarxa, mitjançant la creació d'un sistema on els datagrames enviats al lloc d'origen del node mòbil s'envien a qualsevol localització on es trobi el dispositiu. Per tant, és particularment útil per dispositius sense cablejar, però es pot usar per qualsevol dispositiu que es mogui per les xarxes de forma periòdica.

Com ja s'ha comentat, *Mobile IP* sovint està associat amb xarxes no cablejades, ja que els dispositius que usen tecnologia WLAN es poden moure molt fàcilment d'una xarxa a una altra. Tot i això, no va ser dissenyat específicament per *wireless*. Pot ser igualment útil per passar d'una xarxa *Ethernet* d'un edifici cap a una altra xarxa a un altre edifici, ciutat o país.

És important adonar-se de que *Mobile IP* té certes limitacions a la seva utilitat dins d'un entorn *wireless*. El protocol va ser dissenyat per suportar mobilitat de dispositius, però només una relativa i infreqüent mobilitat. Això es deu a tot el treball involucrat amb cada canvi. És una sobrecàrrega que realment no té un gran impacte quan es mou un dispositiu un cop cada setmana, al dia o fins i tot cada hora. Però si pot ser un problema per mobilitat en "temps real", tal com *roaming* en una xarxa *wireless*, on les funcions operatives de *hand-off* a la capa d'enllaç de dades poden ser més adequades. *Mobile IP* va ser dissenyat sota una específica assumpció de que el punt d'unió no canviaria més d'un cop per segon.

Degut a aquest protocol *Mobile IP*, ni el *Tracker* ni el *Peer 2* són conscients del canvi de xarxa que ha fet el *Peer 1*. Tot i això, degut a l'inherent *tunneling* i enrutament triangular del protocol *Mobile IP* (a través del *Home Agent*) la sessió P2P pot experimentar certa latència en alguns escenaris. Això pot afectar de forma adversa a l'experiència de l'usuari del P2P.

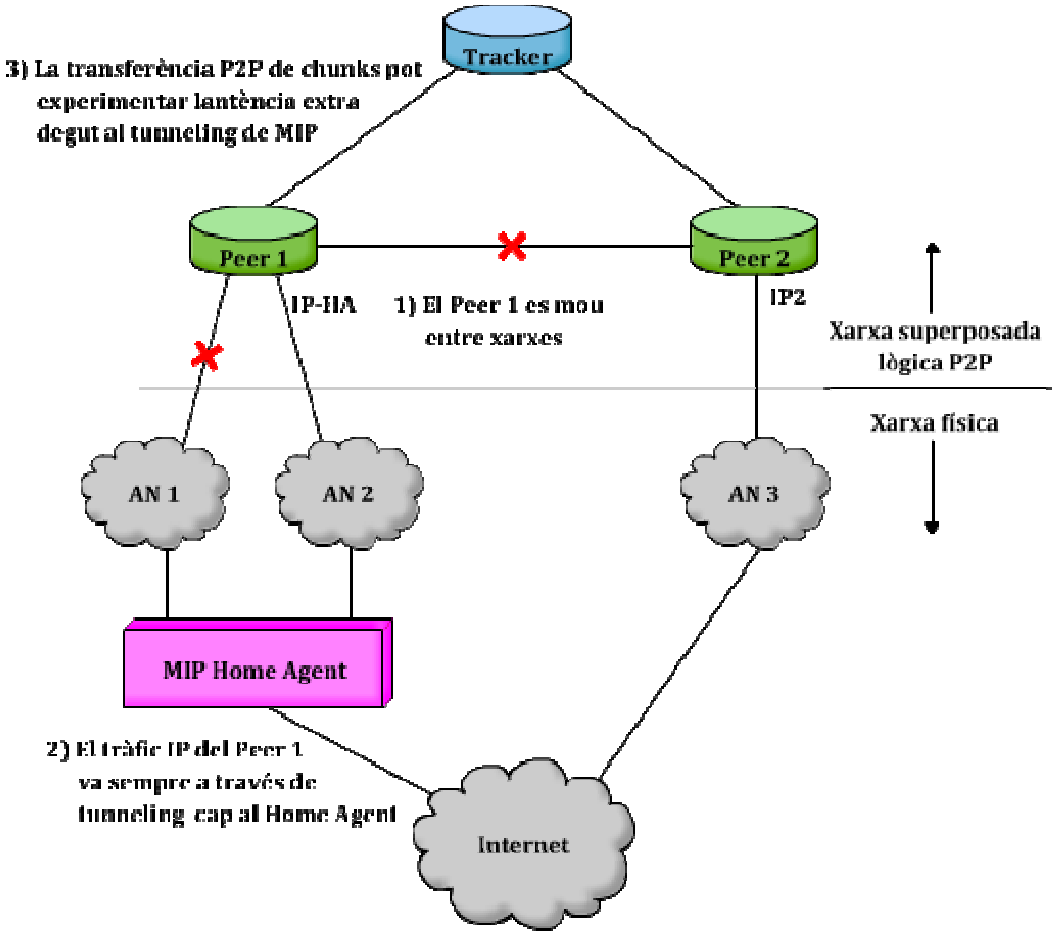

Fig. 6.3.- Streaming P2P amb Mobile IP

6.4.2. Proxy Mobile IP Protocol

Proxy Mobile IPv6 (PMIPv6 o PMIP) [16] és un protocol de gestió de mobilitat basat en xarxa estandarditzat per l'IETF. És un protocol per la creació d'una tecnologia comú i d'accés independent de la xarxa central mòbil, amb capacitat per diverses tecnologies tals com WiMAX, 3GPP o arquitectures d'accés basades en 3GPP i WLAN.

La gestió de mobilitat basada en xarxa habilita les mateixes funcionalitats que *Mobile IP*, sense cap de les modificacions del *protocol stack* de TCP/IP del host. Amb PMIP el host pot canviar el punt d'unió cap a Internet sense necessitat de canviar la seva adreça IP. De forma contrària a l'enfocament de Mobile IP, aquesta funcionalitat s'implementa mitjançant la xarxa, que serà la responsable de rastrejar els diferents moviments del host i d'iniciar la senyalització de mobilitat requerida al seu nom. Tot i així, en cas de que la mobilitat inclogui diferents interfícies de xarxa, el host necessita modificacions similars a *Mobile IP* amb la finalitat de mantenir la mateixa adreça IP a través de diferents interfícies.

L'estàndard del IETF defineix dues entitats, o elements funcionals, de xarxa que estan involucrades a la gestió de mobilitat:

- **Mobile Acces Gateway**, que és una funció on un router d'accés que gestiona la senyalització de mobilitat relacionada per un node mòbil que està unida al seu enllaç d'accés.

- **Local Mobility Anchor**, que és el *Home Agent* pel node mòbil a un domini PMIP amb *Acces Independent Mobile Core*.

En resum, l'ús del *Proxy Mobile IP Protocol* [RFC5213] causa uns problemes i qüestions similars als comentats anteriorment al subapartat de *Mobile IP Protocol* acabat de veure. Deixant de banda aquest fet, *Proxy IP Mobile* també introdueix una nova qüestió a les sessions d'*streaming P2P*. Partint de que el protocol *Proxy Mobile IP* és una solució basada en xarxa, el node mòbil (o *peer*) no és conscient de la seva mobilitat d'IP de forma que no pot informar al *Tracker*, a la *caché* de P2P, a cap CDN o a altres *peers* de mobilitat de nivell IP.

Per tant, la mobilitat IP és totalment invisible per les entitats de sessió d'*streaming P2P* i més difícil de detectar i de respondre en conseqüència. D'aquesta forma *Proxy Mobile IP* tindrà impacte tant en la connexió entre *peers* així com en la de *Tracker* a *peer*.

6.4.3. Host Identity Prototol

Host Identity Protocol (HIP)[17] és una tecnologia d'identificació de host per usar sota xarxes basades en el *Internet Protocol* (IP). Com sabem, Internet consta de dos espais principals de nom, les adreces IP i les DNS. El que fa HIP és separar l'identificador corresponent a l'*end-point* i les funcions de localització de les adreces IP. S'introdueix un host d'identitat (HI), basat en una infraestructura de clau de seguretat pública. El protocol HIP proporciona mecanismes i mètodes de seguretat per *IP multihoming* i computació basada en dispositius mòbils.

A les xarxes que implementen el *Host Identity Protocol*, totes les ocurrències d'adreces IP a les aplicacions són eliminades i reemplaçades amb identificadors de host criptogràfics. Aquestes claus criptogràfiques són típicament autogenerades, tot i que no és estrictament necessari. L'efecte d'aquesta eliminació de les adreces IP a les capes d'aplicació i transport és un desacoblament de la capa de transport respecte de la capa d'interconnexió (capa d'Internet) a TCP/IP.

HIP va ser especificat mitjançant el grup de treball HIP WG del IETF. D'altra banda, un grup de recerca de HIP pertanyent al IRTF[18] s'encarrega de seguir les repercussions més amplies del protocol.

El grup de treball ha sigut creat per produir RFCs sobre el protocol experimental que és HIP, però s'entén que la seva qualitat i les propietats de seguretat han de coincidir amb els estàndards dels requeriments. El propòsit principal per produir documentació *Experimental* en comptes d'estàndards pròpiament dits es deu als efectes desconeguts que els mecanismes poden tenir en algunes aplicacions a Internet a la llarga.

L'especificació del protocol *Host Identity Protocol-Based Overlay Networking Environment* (HIP-BONE)[19] proporciona un *framework* d'alt nivell per la construcció de superposicions basades en HIP. El *framework* de HIP BONE deixa l'especificació dels detalls de com combinar un protocol de *peer* particular amb HIP per construir una plantilla de documents referits com especificacions d'instància de HIP BONE. Una instància d'especificació de HIP BONE definirà, com a mínim:

- El protocol de *peer* que s'usarà.
- Quin tipus d'identificador de nodes (Node ID) s'usarà i com es deriven.
- Quins protocols de *peer* primitius generen els missatges de HIP.
- Com es genera l'identificador de superposició.

7. Implementació pràctica

7.1. Necessitat

La necessitat d'implementar aquesta aplicació surt a partir de voler veure el funcionament de l'intercanvi de missatges del *PPSP Tracker Protocol* entre *Tracker* i *peer*. Hem de tenir present que s'ha partit de l'aplicació programada en C++, fent una adaptació per dispositius mòbils i programada en *Android*. D'aquesta forma, i tenint compatibilitat total, es pot simular un entorn *peer to peer* i veure com es l'intercanvi de missatges, quina informació circula per la xarxa i com es tracta a l'hora de construir les respostes.

Com ja s'ha vist, PPSP és un protocol de senyalització i no de transport, per tant no hi ha intercanvi de fluxos de dades entre nodes, sinó que la comunicació es farà entre node i *Tracker* i serà només per intercanviar missatges de senyalització, ja sigui per gestió de connexió o per enviar informació estadística o d'informació sobre el conjunt.

D'altra banda, per tal de poder veure tota la informació que circula per la xarxa referent al protocol que ens ocupa, es fa servir la definició del *PPSP Tracker Protocol* per *Wireshark*[20] en Lua[21]. És el que es coneix com a *dissector*. Això és necessari degut a que actualment, com el protocol encara no està definit oficialment ni s'utilitza de forma comercial, cap aplicació el fa servir ni el té em compte, i per tant a l'hora de voler fer servir un analitzador de xarxes com *Wireshark*, ens trobem amb que hem de fer una definició pròpia del protocol per poder treballar-hi.

Així doncs, amb tot el conjunt format per les aplicacions, situada en dos o més nodes diferents interconnectats, que poden ser terminals fixes o dispostius mòbils, i *Wireshark* com a analitzador de xarxes amb el *dissector* programat de PPSP, tenim un entorn idoni per provar l'intercanvi de missatges de petició i resposta entre els nodes. D'aquesta forma es pot analitzar cada paquet enviat a través de la xarxa i veure els resultats tant al camp de Log de l'aplicació, com a les captures de *Wireshark*.

Als subapartats que hi ha a continuació, s'explica amb detall la totalitat de l'aplicació desenvolupada en Android per l'intercanvi de missatges (descripció del programa, interfície gràfica, diagrama de blocs, classes, etc.), com els canvis necessaris que s'han fet a l'aplicació programada en C++ i al *dissector* desenvolupat per *Wireshark* basat en PPSP i programant en llenguatge Lua. Finalment, es mostraran els resultats obtinguts a l'escenari complet, acompanyats de captures de pantalla tant de les aplicacions com dels paquets capturats per *Wireshark*.

7.2. Android

7.2.1. Introducció

Android[22] és un sistema operatiu basat en *Linux*[23] per dispositius mòbils, tals com *smartphones* o telèfons intel·ligents i *tablets*. Va ser desenvolupat inicialment per *Android Inc.*, una firma que va ser comprada per *Google*[24] a l'any 2005. És el principal producte de la *Open Handset Alliance*, un conglomerat de fabricants i desenvolupadors de hardware, software i operadors de servei. Les unitats d'*smartphones* amb *Android* s'han ubicat al primer lloc als Estats Units, durant el segon i tercer trimestre de l'any 2010, amb una quota de mercat del 43,6% durant aquest tercer trimestre.

Un dels punts forts dels que fa gala *Android* és la gran comunitat de desenvolupadors que el suporta, programant aplicacions per ampliar la funcionalitat dels dispositius que funcionen amb aquest sistema operatiu. Actualment existeixen més de 200.000 aplicacions disponibles per *Android*. *Android Market* és la botiga virtual online administrada per *Google* des d'on es poden comprar les aplicacions, tot i que existeix la possibilitat d'obtenir software externament. Aquests programes estan escrits amb el llenguatge de programació *Java*[25].

L'anunci del sistema *Android* és va fer el 5 de novembre de 2007 juntament amb la creació de la *Open Handset Alliance*, un consorci de 78 companyies de hardware, software i telecomunicacions dedicades al desenvolupament d'estàndards oberts per dispositius mòbils. *Google* va alliberar la majoria del codi d'*Android* sota la llicència *Apache*[26], una llicència lliure i de codi obert. Actualment *Android* posseeix a prop del 32,9% de la quota de mercat a escala mundial dels smartphones, per davant de *Symbian*[27] que té un 30,6%. En tercer lloc es troba *Apple*[28] amb una quota de mercat del 16%.

L'estructura del sistema operatiu *Android* es composa d'aplicacions que s'executen a dins d'un *framework Java* d'aplicacions orientades a objectes sobre el nucli de les biblioteques Java a una màquina virtual *Dalvik*[29] amb compilació de temps d'execució. Aquestes biblioteques que estan escrites en llenguatge C inclouen un administrador d'interfície gràfica (*surface manager*), un *framework OpenCore*, una base de dades relacional *SQLite*, una API gràfica *OpenGL ES 2.0 3D*, un motor de renderitzat *WebKit*, un motor gràfic SGL[30], SSL[31] i una biblioteca estàndard de C *Glibc*.

Com a curiositat sobre el sistema operatiu, es pot comentar que aquest està composat per 12 milions de línies de codi, incloent 3 milions de línies de XML, 2.8 milions de línies de llenguatge C, 2.1 milions de línies de Java i 1.75 milions de línies de C++.

7.2.2. Breu historia

7.2.2.1. Adquisició per Google

A juliol de 2005, *Google* compra *Android Inc.*, una petita empresa de Palo Alto, Califòrnia, i a partir d'aquí van començar a circular rumors sobre els plans de *Google* de fabricar el seu propi telèfon mòbil lliure, i fins i tot gratuït enfocant-se als guanys de publicitat de les recerques fetes pels usuaris. Finalment aquests rumors d'un telèfon mòbil gratuït van resultar falsos, però *Android* va esdevenir com una cosa molt més interessant i revolucionari: un sistema operatiu mòbil *open source* propulsat ni més ni menys que per la pròpia *Google*.

A *Google* es va començar a desenvolupar aquesta plataforma per dispositius mòbils basada al *kernel* de *Linux*, que va ser promocionat com a fabricants de dispositius i operadors amb la promesa de proveir un sistema flexible i actualitzable. Es va informar de que *Google* havia alineat ja una sèrie de fabricants de hardware i software i que va senyalar als operadors que estava obert a diversos graus de cooperació per la seva part.

L'especulació sobre que el sistema *Android* de *Google* entraria al mercat de la telefonia mòbil es va incrementar cap al desembre de 2006. Diversos reports de la BBC i de *The Wall Street Journal* van senyalar que *Google* volia els seus serveis de recerca i aplicacions a telèfons mòbils i estava molt obstinat amb aquest tema. A partir d'aquí, tant mitjans impresos com publicacions online van informar aviat de que *Google* estava desenvolupant un telèfon amb la seva marca.

Durant setembre de 2007, "*Information Week*" va difondre un estudi de *Evalueserve* que reportava que *Google* havia sol·licitat diverses patents a l'àrea de la telefonia mòbil. El llançament inicial del *Android Software Development Kit* va aparèixer durant novembre de l'any 2007 i molt de temps després, a mitjans d'agost de 2008, va aparèixer el *Android 0.9 SDK* en forma de beta. Un mes després, cap a finals de setembre de 2008, finalment es va lliurar *Android 1.0 SDK (Release 1)*.

7.2.2.2. Open Handset Alliance

Open Handset Alliance (OHA) [32] és una aliança comercial de 78 companyies per desenvolupar estàndards oberts per dispositius mòbils. Alguns dels membres que la

formen són: *Google*, *HTC*, *Dell*, *Intel*, *Motorola*, *Qualcomm*, *Texas Instruments*, *Samsung*, *LG*, *T-Mobile* i *Wind River Systems*.

La OHA és va establir el 5 de novembre de 2007, liderada per *Google* amb altres 34 membres entre els quals s'incloïen fabricants de dispositius mòbils, desenvolupadors d'aplicacions, alguns operadors de comunicacions i fabricants de xips. *Android*, el software estrella de l'aliança, està basat en una llicència open source i competeix directament contra plataformes mòbils com *Apple*, *Microsoft*, *Nokia*, *Palm*, *Research In Motion*, *Symbian* i *Bada*.

Al mateix temps que s'anunciava la formació de la OHA, aquesta va presentar *Android*, la seva plataforma de codi lliure per mòbils basada en *Linux*. Una beta per desenvolupadors es va lliurar uns dies després, el 12 de novembre.

El primer telèfon comercialment disponible amb *Android* va ser el *T-Mobile G1*, conegut també com *HTC Dream*. Va ser aprovat per la FCC[33] el 18 d'agost de 2008 i va sortir a la venda el 22 d'octubre.

7.2.2.3. Llicències

Amb l'excepció de curts períodes d'actualització, *Android* ha estat disponible sota una llicencia *open source* de software lliure des del 21 d'octubre de l'any 2008. *Google* va publicar tot el codi font (incloent els *stacks* de xarxa i telefonia), sota una llicència *Apache*. A més a més, *Google* manté la llista de temes revisats oberta per qualsevol usuari que desitgi veure-la i comentar-la.

7.2.2.4. Historial de versions

L'historial de versions del sistema operatiu Android va començar amb la release de la versió 1.0 del sistema al setembre de 2008. Android ha anat rebent nombroses actualitzacions des de que va sortir la seva primera versió. Aquestes actualitzacions del sistema operatiu que fa de base, són típicament per tal de solucionar problemes o *bugs*, i afegir noves funcionalitats. Generalment cadascuna de les noves versions desenvolupades apareix sota un nom clau basat en postres culinaris. Aquests noms claus estan en ordre alfabètic.

Les versions d'Android lliurades fins avui són:

- *Android 1.0*, l'original, lliurada el 23 de setembre de 2008.

- *Android 1.1*, lliurada el 9 de febrer de 2009, amb diverses característiques afegides tot i que només estava disponible per T-Mobile G1.

- *Android 1.5*, anomenada *Cupcake* (magdalena) i lliurada el 30 d'abril de 2009.

- *Android 1.6*, coneguda com *Donut* i lliurada el 15 de setembre de 2009.

- *Android 2.0/2.1*, coneguda com *Eclair* (unes postres franceses) i lliurada el 26 d'octubre de 2009, introduint suport per HTML5 i per *Exchange ActiveSync 2.5*.

- *Android 2.2*, anomenada *Froyo* (de *Frozen Yogurt* o iogurt gelat) i lliurada el 20 de maig de 2010, introduïa millores de velocitat amb optimització JIT i el motor *JavaScript Chrome V8*, a més d'afegir suport per *Adobe Flash* i immobilització de punt d'accés via WiFi.

- *Android 2.3*, coneguda com *Gingerbread* (pa de gingebre) i lliurada el 6 de desembre de 2010, va redefinir l'interfície d'usuari, millorar el teclat virtual i característiques de *copy/paste*, a més d'afegir suport per *Near Field Communication*[34].

- *Android 3.0*, anomenada *Honeycomb* (bresca), és una versió d'Android orientada només als dispositius *tablet*. Aquesta versió va ser lliurada el 26 de gener de 2011 i entre d'altres novetats, suporta dispositius amb pantalles molt grans així com el fet d'encarregar-se d'introduir noves característiques per l'interfície d'usuari i suportar processadors *multicore* i acceleració de hardware per gràfics.

- *Android 4.0*, anomenada *Ice-cream* (gelat) serà lliurada segurament a mitjans del present any 2011.

7.2.3. Arquitectura

Els components principals que conformen l'arquitectura del sistema operatiu d'*Android* són, descrits amb detall, els següents:

- **Aplicacions:** les aplicacions base inclouen un client de correu electrònic, programa d'SMS, calendari, mapes, navegador i contactes entre d'altres. Totes les aplicacions estan escrites en llenguatge de programació *Java*.

- **Marc de treball d'aplicacions:** els desenvolupadors tenen accés complet a les mateixes APIs del *framework* usades per les aplicacions base. L'arquitectura està desenvolupada per simplificar la reutilització de components. És a dir, qualsevol

aplicació pot publicar les seves capacitats i qualsevol altra aplicació pot després utilitzar aquestes capacitats. Capacitats subjectes a les regles de seguretat del *framework*. Aquest mateix mecanisme permet que els components siguin substituïts per l'usuari.

- **Biblioteques:** *Android* inclou un conjunt de biblioteques de C/C++ utilitzades per diversos components del sistema. Aquestes característiques s'exposen als desenvolupadors a través del marc de treball d'aplicacions d'*Android*. Algunes d'aquestes són: *System C library* (implementació de biblioteca C estàndard), biblioteques de medis, biblioteques de gràfics, 3D i *SQLite*, entre d'altres.

- **Runtime d'Android:** *Android* inclou un set de biblioteques base que proporcionen la major part de les funcions disponibles a les biblioteques base del llenguatge *Java*. Cada aplicació *Android* corre el seu propi procés, amb la seva pròpia instància de la màquina virtual *Dalvik*. *Dalvik* ha sigut escrit de forma que un dispositiu pot córrer múltiples màquines virtuals de forma eficient. *Dalvik* executa arxius en el format *Dalvik Executable* (.dex), que està optimitzat per memòria mínima. La *Virtual Machine* està basada en registres i pot córrer classes compilades pel compilador de *Java* que han sigut transformades al format ".dex" per l'eina inclosa "dx".

- **Nucli Linux:** *Android* depèn de *Linux* pels serveis base del sistema tals com seguretat, gestió de memòria, gestió de processos, pila de xarxa i model de controladors. El nucli també actua com una capa d'abstracció entre el hardware i la resta de la pila de software.

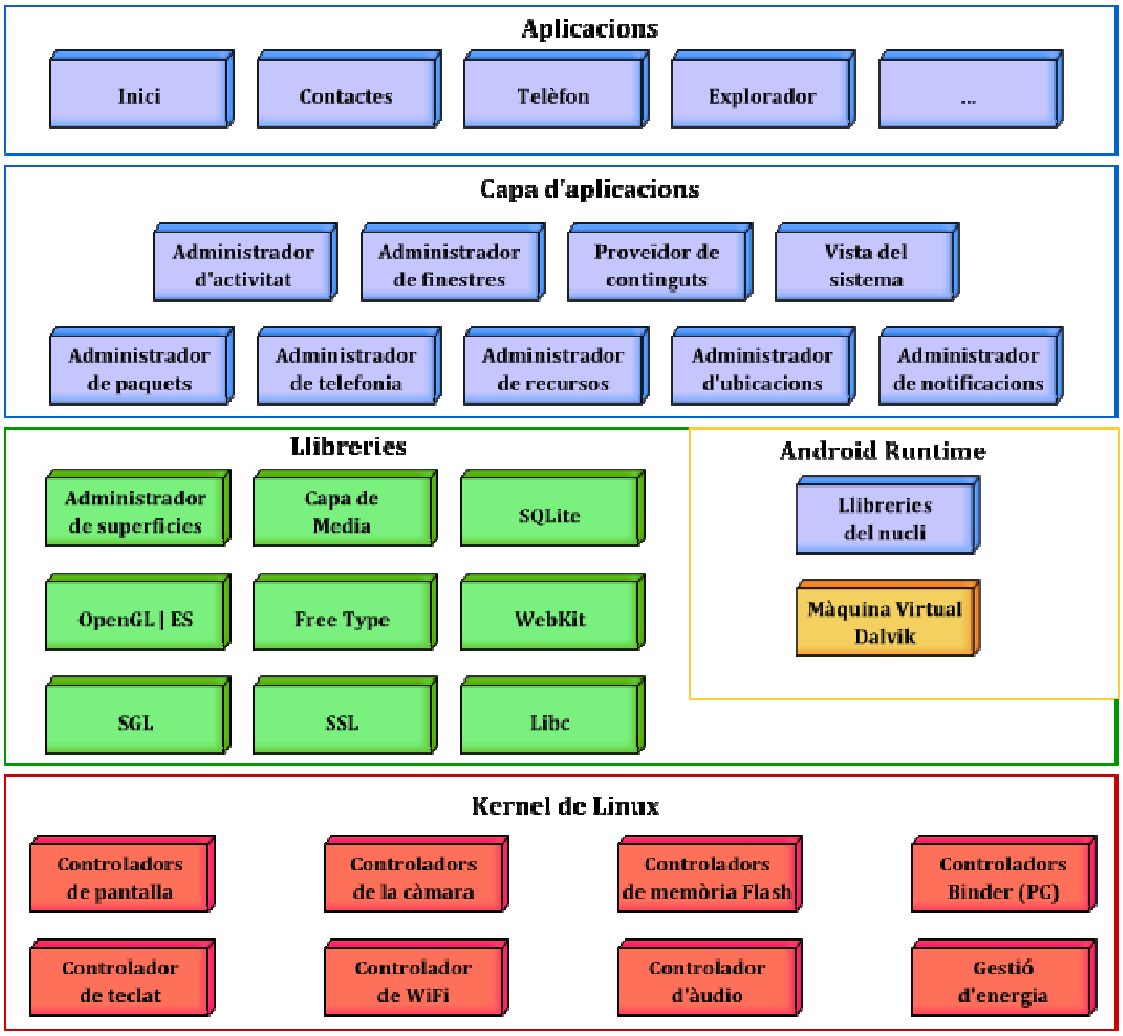

Fig. 7.1.- Diagrama de l'arquitectura d'Android

7.2.4. Característiques

Les característiques i especificacions actuals d'*Android* inclouen:

- **Disseny de dispositiu:** La plataforma és adaptable a pantalles més grans, VGA, biblioteca de gràfics 2D, biblioteca de gràfics 3D basada a les especificacions de la *OpenGL ES 2.0* i disseny de telèfons tradicionals.

- **Emmagatzemament:** *SQLite*, una base de dades lleugera, la qual s'usa per propòsits d'emmagatzemament de dades.

- **Connectivitat:** Android suporta les següents tecnologies de connectivitat: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, *Bluetooth*, Wi-Fi, LTE i WiMAX.

- **Missatgeria:** SMS i MMS són dues formes de missatgeria, incloent missatgeria de text i ara l'*Android Cloud to Device Messaging Framework*.

- **Navegador web:** El navegador de web inclòs a *Android* està basat al motor de renderitzat de codi obert *WebKit*, emparellat amb el motor *JavaScript V8* de *Google Chrome*.

- **Suport de Java:** Tot i que les aplicacions estan escrites en *Java*, no hi ha una màquina virtual de *Java* a la plataforma. El codi *Java* no s'executa, sinó que es compila a l'executable *Dalvik* i corre a la *Dalvik Virtual Machine*. *Dalvik* és una màquina virtual especialitzada dissenyada específicament per *Android* i optimitzada per dispositius mòbils que funcionen amb bateria i que tenen memòria i processador limitats. El suport per J2ME pot ser agregat mitjançant aplicacions de tercers tals com el *J2ME MIDP Runner*.

- **Suport multimèdia:** *Android* suporta els següents formats multimèdia: WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB (amb un contenidor 3GP), AAC, HE-AAC (amb contenidors MP4 o 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG i BMP.

- **Suport per streaming:** *Streaming RTP/RTSP* (3GPP PSS, ISMA), descàrrega progressiva de HTML (HTML5 <video> tag). *Adobe Flash Streaming* (RTMP) es suportat mitjançant el *Adobe Flash Player*. Es planeja el suport de *Microsoft Smooth Streaming* amb el port de *Silverlight* a *Android*. *Adobe Flash HTTP Dynamic Streaming* estarà disponible mitjançant una actualització de *Adobe Flash Player*.

- **Suport per hardware addicional:** *Android* suporta càmeres de fotos, de vídeo, pantalles tàctils, GPS, acceleròmetres, giroscopis, magnetòmetres, sensors de proximitat i de pressió, termòmetre, acceleració 2D i 3D.

- **Entorn de desenvolupament:** Inclou un emulador de dispositius, eines per la depuració de memòria i l'anàlisi del rendiment del software. L'entorn de desenvolupament integrat és *Eclipse* usant el *plugin* d'Eines de Desenvolupament d'*Android*.

- **Market:** L'*Android Market* és un catàleg d'aplicacions que poden ser descarregades i instal·lades a dispositius *Android* sense la necessitat d'un PC.

- **Multitàctil:** *Android* té suport natiu per pantalles multitàctils que inicialment van fer la seva aparició a dispositius com l'*HTC Hero*.

- **Bluetooth:** El suport per A2DF i AVRCP va ser agregat a la versió 1.5. L'enviament d'arxius (OPP) i l'exploració del directori telefònic a la versió 2.0, i la marcació per veu juntament amb l'enviament de contactes entre telèfons a la 2.2.

- **Videotrucada:** La versió principal d'*Android* no suporta videotrucada, tot i això alguns dispositius podrien tenir una versió personalitzada del sistema operatiu que si ho suporta, ja sigui mitjançant la xarxa de l'operador o sobre IP.

- **Multitasca:** La multitasca real d'aplicacions està disponible.

- **Característiques basades en veu:** La recerca a *Google* a través de veu està disponible com "Entrada de recerca" des de la versió inicial del sistema.

- **Tethering:** *Android* suporta *tethering*, que permet al telèfon ser usat com un punt d'accés cablejat o sense cablejar.

7.2.5. Desenvolupament

Android, al contrari que altres sistemes operatius per dispositius mòbils com iOS[35] o *Windows Phone*[36], es desenvolupa de forma oberta i es pot accedir tant al codi font com al llistat d'incidències on es poden veure problemes encara no resolts i reportar problemes nous.

El fet de que es pugui tenir accés al codi font no significa necessariament que es pugui tenir sempre l'última versió d'*Android* a un determinat mòbil, degut a que el codi per suportar el hardware de cada fabricant normalment no és públic, així que faltaria una part bàsica del *firmware* per poder fer-lo funcionar al terminal particular, i perquè les noves versions d'*Android* generalment solen demanar més recursos, pel que els models més antics queden descartats per raons de memòria RAM i velocitat de processador, entre d'altres.

A l'actualitat existeixen més de 200.000 aplicacions per *Android* i s'estima que sobre 300.000 telèfons mòbils s'activen diàriament.

La botiga d'aplicacions d'*Android*, coneguda com *Android Market*, retribueix als desenvolupadors el 70% del preu de la seva aplicació. Així mateix, el desenvolupament d'aplicacions per *Android* no requereix aprendre llenguatges complexes de programació. Tot el que es necessita es un coneixement acceptable de *Java* i disposar del *Software Development Kit* proveït per *Google*, que es pot descarregar gratuïtament.

7.3. Socket entre C++ i Java

Una part important de la part pràctica del projecte, cau en la comunicació entre un node programat sota Android amb Java i l'aplicació del *Tracker*, programada en C++.

Connectar una aplicació programada en Java amb una en C++ mitjançant *sockets* no és trivial i pot comportar certs problemes a l'hora de fer la implementació. Problemes que ens podem trobar a l'hora de la connexió i de transmetre dades.

Aquests problemes són habituals quan connectem mitjançant *sockets* programes que corren a plataformes distintes, sobretot tenint en compte que les aplicacions Java corren sota la seva pròpia màquina virtual.

Abans de veure els problemes i les solucions pertinents per tal d'establir una comunicació mitjançant sockets entre dues aplicacions programades en C++ i Java, és interesant fer una petita introducció, per sobre, d'ambós llenguatges de programació.

7.3.1. C++

C++ és un llenguatge de programació dissenyat a mitjans dels anys 80 per Bjarne Strostrup. L'intenció que es buscava amb la seva creació va ser la d'estrendre l'exitós llenguatge de programació conegut com C, amb mecanismes que permetessin la manipulació d'objectes. En aquest sentit, des del punt de vista dels llenguatges orientats a objectes, el C++ és un llenguatge híbrid.

Posteriorment, es van afegir facilitats de programació genèrica, que es va sumar als altres dos paradigmes que ja estaven admesos (concretament la programació estructurada i la programació orientada a objectes). Per això es sol dir que C++ és un llenguatge de programació multiparadigma.

Actualment existeix un estàndard, denominat ISO C++, al que s'han afegit la majoria dels fabricants de compiladors més moderns. Existeixen també alguns intèrprets, tals com ROOT.

Una particularitat del C++ és la possibilitat de redefinir els operadors, o sobrecàrrega d'operadors, i de poder crear nous tipus que es comporten com a tipus fonamentals.

El nom de C++ va ser proposat per Rick Mascitti l'any 1983, quan el llenguatge va ser usat per primer cop fora d'un laboratori científic. Abans s'havia usat el nom de "C amb classes". A C++, l'expressió "C++" significa "increment de C" i es refereix a que C++ és una extensió de C.

7.3.2. Java

Java és un llenguatge de programació orientat a objectes, desenvolupat per *Sun Microsystems*[37] a principis dels anys 90. El llenguatge en si mateix agafa molta sintaxi de C i C++, però té un model d'objectes més simple i elimina eines de baix nivell, que generalment indueixen a errors, com la manipulació directa de punters o memòria.

Les aplicacions *Java* estan típicament compilades en un *bytecode*, tot i que la compilació en codi màquina natiu es també possible. Durant el temps d'execució, el *bytecode* és normalment interpretat o compilat a codi natiu per l'execució, tot i que l'execució directa per hardware del *bytecode* per un processador *Java* és possible.

La implementació original i de referència del compilador, la màquina virtual i les biblioteques de classes de Java van ser desenvolupades per *Sun Microsystems* l'any 1995. Des de llavors, *Sun* ha controlat les especificacions, el desenvolupament i l'evolució del llenguatge a través del *Java Community Process*, tot i que altres han desenvolupat també implementacions alternatives d'aquestes tecnologies de *Sun*, algunes fins i tot sota llicències de software lliure.

Entre desembre de 2006 i maig de 2007, *Sun Microsystems* va alliberar la major part de les seves tecnologies Java sota la llicència *GNU GPL*[38], d'acord amb les especificacions del *Java Community Process*, de tal forma que pràcticament tot el *Java* de *Sun* actualment és software lliure (tot i que la biblioteca de classes de *Sun* que es requereix per executar els programes *Java* encara no ho és).

7.3.3. Algunes diferències entre plataformes

Al mercat hi ha una gran quantitat de microprocessadors i cada ordinador decideix quin s'utilitza. Els PC solen usar *Pentium* o compatibles, les estacions de treball SUN tenen un micro diferent (*Sparc*), els *Mac* un altre (*PowerPC*), i així un llarg etc.

El problema és que cada micro d'aquests defineix els enters, *chars* i la resta de tipus de variable d'una forma diferent. El normal és que un enter, per exemple, siguin quatre bytes, tot i que en alguns micros antics eren de dos bytes i els més moderns ja comencen a ser de vuit.

Dins dels quatre bytes dels que parlem, els Pentium fan que el byte menys significatiu ocupi l'adreça més baixa de memòria, mentre que els Sparc ho fan al revés. Aquestes representacions són les que es coneixen com *little endian*[39] (*Intel 80x86*, *Dec Vax* i *Dec Alpha*) i *big endian*[40] (IBM 360/370, *Motorola*, *Sparc*, HP PA i la màquina virtual de *Java*), respectivament.

D'aquesta forma, si enviem un enter a través d'un *socket*, estem enviant aquests quatre bytes. Si els micros a ambdós costats del socket tenen el mateix ordre pels bytes no hi haurà problema, però si es diferent, s'interpretarà incorrectament.

La màquina virtual Java defineix els *char* de dos bytes per tal de poder usar caràcters UNICODE, mentre que en la resta de microcontroladors habituals un char sol ser d'un únic byte. Si enviem des de Java un caràcter a través d'un socket, enviem dos bytes, mentre que si el llegim del socket mitjançant un programa en C, només llegirem un byte i es deixarà a l'altre pendent de lectura.

En qüestió de *floats* i *doubles*, la cosa es complica i no és tan simple com invertir quatre *bytes*.

7.3.4. Format estàndard de xarxa

La solució a aquests problemes passa per enviar les dades d'una forma més o menys estàndard, independentment del microcontrolador, per tal de que es puguin entendre sense massa problemes.

Tal qual circulen per Internet, els enters són de quatre *bytes* i van ordenats de la mateixa forma que en *Java* o *Sparc*. D'altra banda, els *chars* són d'un *byte*.

Si estem en un *Pentium*, abans d'enviar un enter per un *socket*, cal fer un codi que serà necessari per poder invertir l'ordre dels *bytes*. Evidentment, aquesta inversió s'ha de fer també a la recepció al llegir *bytes*. Però aquest codi només valdrà a un *Pentium*. Si el codi font s'executés a un *Linux* que corri a un processador *Sparc*, aquesta part de codi que inverteix els *bytes* s'hauria d'eliminar.

Afortunadament, tant en C com de linux com de Windows (gràcies a winsocket), tenim la familia de funcions htonl( ).

- **htonl ( )** passa un enter de format *hardware* (el del microprocessador) a format de xarxa (*Hardware TO Network*).
- **ntohl ( )** passa un enter de format de xarxa a format *hardware*.
- **htons ( )** fa el mateix que *htonl ( )*, però amb un *short* (un enter de dos *bytes* en comptes de quatre).
- **ntohs ( )** fa el mateix que *ntohl ( )*, però amb un *short*.

Aquestes diferents funcions estan implementades per cada microcontrolador en concret, fent el que sigui necessari en cada cas. D'aquesta forma, si abans d'enviar un enter per un *socket* cridem a la funció *htonl ( )* i després de llegir-lo cridem a *ntohl ( )*, l'enter circularà per la xarxa a un format estàndard i qualsevol programa que el tingui en compte serà capaç de llegir-lo.

Cridant a aquestes funcions el nostre codi font és a més portable d'una màquina a una altra, evitant això que s'hagi de modificar el codi. Simplement bastarà amb recompilar el codi per cadascuna de les màquines. A un *Pentium* aquestes funcions invertiran els *bytes* mentre que a un *Sparc* no fan res, però existeixen i compilen sense errors.

En quant als char, degut a que *Java* és l'únic que de moment utilitza dos *bytes*, en cas de voler enviar-ne, una opció seria que el mateix codi *Java* converteixi aquests dos caràcters en un únic *byte* abans d'enviar i els reconverteixi a dos a l'hora de rebre. A la classe *String* de *Java* hi ha mètodes que permeten fer això.

7.4. Descripció de l'aplicació

7.4.1. Introducció

Fent una petita introducció, el que es pretén amb aquesta aplicació és fer una simulació de l'entorn *peer to peer* en el que ha de treballar el protocol PPSP. En aquest cas, s'ha desenvolupat una aplicació en *Android* que actuarà com a *peer* i podrà interactuar amb l'aplicació desenvolupada en C++ anteriorment, que pot ser *Tracker* o *peer*. Per tal d'aconseguir-ho, aquest entorn pot ser de dues formes.

El que es pretén és emular un entorn *peer to peer* centralitzat. Un node situat en un PC farà de *Tracker*, previ haver-ho indicat a l'opció corresponent de la configuració; i la resta, ja siguin nodes mòbils o nodes fixes, de *peers*. Aquest *Tracker* centralitzat serà l'encarregat de rebre tots els missatges de petició que facin els peers i respondre segons convingui.

7.4.2. Diagrama d'entitats

Abans d'entrar en detall amb la implementació de l'aplicació a nivell d'estructura i de codi, cal mostrar primer el diagrames de blocs amb les diferents entitats que formen el sistema *peer to peer* que es vol muntar. Hem de tenir present que el diagrama que veurem, inclou tant nodes funcionant amb el *PPSP Node Communication* (l'aplicació programada en C++) funcionant com a *Tracker* o com a *peer*, i nodes funcionant amb el *PPSP Droid Node* (l'aplicació programada en *Android*).

Aquest diagrama serà, tal com veiem a continuació, un sistema *peer to peer* amb *Tracker* centralitzat:

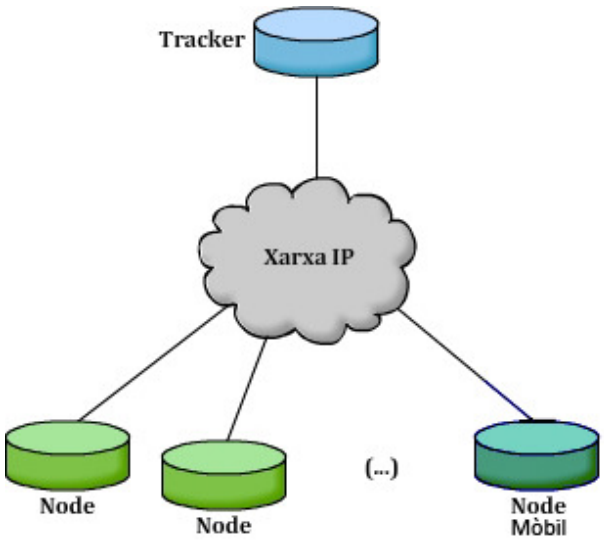

Fig. 7.2.- Diagrama de blocs amb diferents nodes

Es pot observar un grup de nodes connectats a la xarxa i amb un Tracker, que es qui gestionaria les dades i les connexions. Com s'ha comentat, un dels nodes actuarà només com a *Tracker*, havent-ho indicat així a la configuració, i la resta de nodes actuaran com a *peers*; sent alguns nodes mòbils funcionant sota Android i amb l'aplicació corresponent. D'aquesta forma, s'emula el funcionament d'una xarxa *peer to peer* centralitzada.

Per a completar l'escenari, a l'esquema se li ha d'afegir una màquina, que pot ser també qualsevol dels nodes, que tingui funcionant *Wireshark* capturant els paquets que circulen per la xarxa per tal d'analitzar-los i així poder visualitzar els diferents missatges de petició i de resposta que es van generant i enviant.

Per tal de filtrar només els paquets corresponents al protocol PPSP, es fa servir un *dissector* del protocol com ja s'ha comentat, desenvolupat expressament per visualitzar els paquets del protocol PPSP.

7.4.3. Esquema de l'estructura i descripció de classes

Un cop vist com serà la xarxa que es muntarà, podem aprofundir més en l'aplicació si ens centrem a l'estructura interna de la mateixa, desglossant l'aplicació en les diferents classes que la conformen.

Per motius d'encapsulació i d'haver-se desenvolupat en *Java*, que és un llenguatge de programació orientat a objectes, la programació de l'aplicació s'ha dividit en diverses classes segons la funcionalitat de cadascun dels diferents blocs que l'havien de conformar. En aquest apartat hi dedicarem unes línies a cada una d'aquestes classes per tal d'explicar part per part que fan els diferents blocs que integren l'aplicació, així com les funcions i les variables més representatives que les conformen.

Primer, però, veurem l'esquema de l'estructura de les classes que conformen l'aplicació, de forma que es poden apreciar les diferents classes existents i com estan connectades entre elles. És el que segueix a continuació:

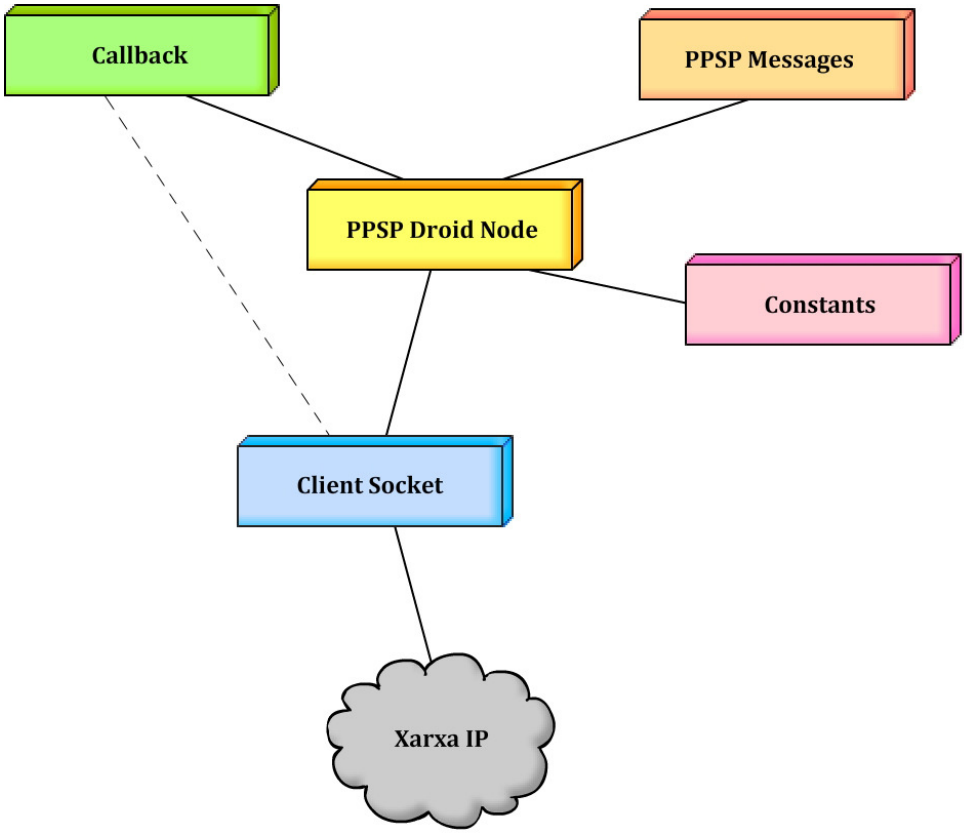

Fig. 7.3.- Esquema de l'estructura de classes

Seguint l'estructura que segueix el programa al inicialitzar-se, aquest comença per la classe *PPSPDroidNode*, que és la classe encarregada de crear els diferents diàlegs, juntament amb la interfície gràfica en general, i la és la classe responsable de tota la gestió de l'aplicació i de la creació de la resta d'objectes de les altres classes. És a dir, les classes que implementen el *socket* i l'objecte de la classe dels missatges PPSP, responsable de la generació dels diferents tipus de missatges del protocol. A més de la llibreria que gestiona les constants i el *Callback*, que es comentarà més endavant. *PPSPDroidNode* és, bàsicament, el centre neuràlgic de l'aplicació.

El primer que fa aquesta classe és inicialitzar totes les variables de configuració inicials que conformen el programa. Juntament amb la inicialització de tots els paràmetres, es crea un objecte de la classe *ClientSocket* i un altre de la classe *PPSPMessages*, a més del l'objecte de la classe *Callback*. La primera d'aquestes dues classes correspon a una classe que ens proporciona un *socket* que té la funció d'estar contínuament escoltant, és a dir, esperant una petició de connexió per part d'un altre node, al port corresponent. A més, fa servir aquest *socket* per enviar els diferents missatges del protocol que es formaran, així com els diferents missatges de gestió de connexió de l'aplicació en sí.

La segona d'aquestes dues classes és la classe que s'encarregarà de crear els diferents tipus de missatges referents al protocol PPSP, segons els diferents paràmetres que rebi. Aquest objecte estarà gestionat per la classe *PPSPDroidNode* i no serà fins que s'estableixi una connexió amb un node *Tracker* muntat sobre un PC amb l'aplicació desenvolupada en C++, quan es podrà fer servir els mètodes d'aquesta classe (estarà deshabilitat fins llavors).

És en aquest instant, un cop el node hagi establit una connexió amb el *Tracker*, és quan es crearà un objecte de la classe *ClientSocket*, que com ja hem dit serà la classe encarregada d'enviar els diferents missatges a través de la xarxa d'un node a un altre.

Finalment, queda per comentar la classe anomenada *Callback*, que té la missió de fer que es pugui modificar l'interfície gràfica des de classes que no siguin la classe principal; i les *Constants*, on es recullen diferents constants que fan referència al protocol.

Arribats a aquest punt, el que hem vist fins ara ha sigut una visió general de les classes. És interessant aprofundir més en aquestes, comentant-les una a una:

7.4.3.1. PPSPDroidNode

És la classe principal del programa. És la base de l'aplicació i s'encarrega de crear els diferents diàlegs que conformen la interfície gràfica del programa, a més de conformar el que serà el nucli de l'aplicació.

Entre els diferents botons de la interfície implementats a la classe, trobem els dos que serveixen per establir la connexió i desconnexió del node, així com el que serveix per actualitzar els paràmetres de configuració del node. És una forma de resetejar el programa sense necessitat de tenir que tancar i tornar a obrir l'aplicació per recarregar totes les dades de la configuració inicial. Tot això es pot fer a través del botó de menú que incorporen els dispositius que funcionen amb *Android*, on a més, es mostra la informació (versió, data, etc) de l'aplicació i el botó per sortir, tancant tot.

Com ja s'ha comentat, aquesta classe és també la que conforma el nucli de l'aplicació i des d'on es creen els diferents objectes de la resta de classes del programa: *ClientSocket*, *PPSPMessages* i *Callback*. A més, és on hi ha les dades de la configuració del node (Peer ID, ID de l'eixam al que volem connectar, etc.).

Aquestes dades que es llegeixen des d'una base de dades inicial de l'aplicació, guardades a variables globals dins de la classe que es faran servir al llarg del programa per diferents funcions. A més, aquestes es podran modificar o actualitzar en qualsevol

moment, modificant els diversos paràmetres corresponents i actualitzant les variables de l'aplicació amb el botó corresponent de la interfície del menú.

Tenint clara tota la part d'inicialització del programa en general, i de la classe *PPSPDroidNode* en particular, podem centrar-nos en detallar el procés que segueix la classe des de que comença fins que es destrueix un cop es tanca el programa.

Aquest procés comença amb tota la inicialització, de la que ja s'ha parlat, i amb la creació d'un objecte de les classes *Callback*, per tal de poder modificar la interfície des de la resta de classes de l'aplicació; i *PPSPMessage*, que serà la classe encarregada de generar els diferents tipus de missatges definits pel *PPSP Tracker Protocol*. Es podria dir que l'aplicació en aquest punt està en espera d'enviar una petició de connexió, un cop s'hagi connectat amb el *Tracker*. Aquesta petició podrà ser denegada, rebutjada o acceptada, segons si s'han posat de forma correcta els paràmetres de connexió (la IP del node remot), o si el node està disposat a connectar o no.

Un cop s'ha establit aquesta connexió amb un altre node, amb el que farem la connexió *peer-Tracker* de la xarxa P2P, ha entrat en joc la classe *ClientSocket*, que és l'encarregada de gestionar aquesta connexió. És el canal que mantindrà la comunicació entre els nodes, tant per transmissió com per recepció de missatges.

Amb la connexió establida, els nodes poden intercanviar missatges triant una de les opcions del menú de la interfície (controlat a la classe *PPSPDroidNode*). Un cop triada l'opció i pres el botó d'enviar, es generarà el missatge de petició corresponent amb les dades que té el node i l'opció triada, usant la classe *PPSPMessages*, encarregada de generar el missatge a enviar. Havent construït la capçalera i el cos de missatge, s'ajunten en un únic *buffer* i a través de la classe *ClientSocket* es procedeix a fer l'enviament mitjançant la xarxa IP.

A partir d'aquí, aquest node rebrà, de l'altre node, que farà servir l'aplicació en C++ per implementar el *Tracker*, la resposta que correspongui segons el tipus de dades enviades i les característiques configurades al *Tracker*. Aquest, al rebre el missatge de petició del primer node, el processa i recull les dades per avaluar quina ha de ser la resposta a retornar i la construeix, de la mateixa forma que es construiria un missatge de petició, tenint en compte que ara serà un missatge de resposta.

Quan l'intercanvi de missatges de petició i resposta ha finalitzat, l'aplicació torna al mateix estat en que es trobava al haver establit la connexió i a l'espera de rebre més ordres, per enviar o rebre missatges. També, en aquest moment, la informació

recollida a la petita base de dades que conté el *Tracker* en forma d'estructures dinàmiques, si es dona el cas de que sigui necessari, s'haurà actualitzat.

Cal tenir present també, que el port que s'ha fet servir pels *sockets* no és un port que s'hagi definit oficialment pel IETF, ja que encara no han assignat cap port al protocol PPSP. Per tant, degut a això, s'ha agafat un port arbitrari dels que estan lliures, és a dir, que no faci servir cap altre aplicació o protocol. El port triat en qüestió és el 29483.

7.4.3.2. PPSPMessages

Aquesta classe és la classe que s'encarrega de construir, a partir de les dades que rebi des de la classe *PPSPDroidNode*, els diferents missatges del *PPSP Tracker Protocol*. Degut a que aquests diferents missatges tenen tots unes característiques comunes a tots ells, com és la capçalera general de missatge; i unes diferenciatives, el cos de missatge, s'ha optat per separar ambdues parts. D'aquesta manera, la capçalera es forma a partir d'una funció general que s'haurà de cridar per tots els missatges independentment del *Method* que els hi correspongui.

Un cop formada la capçalera, segons el tipus de missatge a formar, es cridarà la funció corresponent i es passaran les dades corresponents. Els missatges de petició tenen tots un cos de missatge corresponent, en canvi, amb els missatges de resposta no passa això, ja que la majoria d'ells no tenen cos de missatge. Degut a això, només hi ha dos tipus de missatge de resposta amb cos de missatge i són aquests dos els únics afegits com a tals a la classe.

El valor afegit d'aquesta classe és que pot funcionar independentment de l'aplicació per usar-la a qualsevol altra aplicació basada en PPSP per crear els missatges del *PPSP Tracker Protocol*. Es pot trobar la classe completa a l'annex del final de la memoria.

7.4.3.3. ClientSocket

ClientSocket és la classe encarregada de les connexions amb la xarxa. És el canal que manté la connexió entre els nodes, permetent transmetre i rebre missatges. En definitiva, *ClientSocket* és la classe que gestiona les connexions.

Des de *PPSPDroidNode*, un cop generats els missatges amb *PPSPMessages*, s'enviaran aquests missatges per la xarxa amb la funció *write( )*, pertanyent a l'*OutputStream* que enviem, que és una funció membre de la classe genèrica *CAsyncSocket*.

D'altra banda, i amb la connexió establida, a la funció *run( )* de *CClientSocket*, que indica que es un *thread* que implementa *Runnable*, està corrent constantment mentre

el *socket* existeixi, de forma que escolta totes les dades que pugui arribar al node, es tracten les dades rebudes a un missatge i es recull tota la informació per tal de generar una resposta adequada. La recepció és fa amb un *BufferedReader*, que agafa les dades d'un *InputStream*.

Amb aquestes dues funcions de la classe *ClientSocket* es tenen controlats els dos camins que poden prendre els missatges, entrada o sortida.

7.4.3.4. Callback

La classe *Callback* és una petita classe que té la missió de que qualsevol classe que la implementi com a objecte pugui modificar la interfície gràfica de la classe principal. En cas de no implementar-la, no es podrà modificar la interfície gràfica.

```
public interface Callback {
    public void returnServiceResponse();

}
```

7.4.4. Interfície gràfica

En aquest sub-apartat es veurà el disseny i les funcions de la interfície gràfica de l'aplicació per Android. Aquesta consta de diverses subpantalles per poder configurar les diferents opcions. Primer, comencem per veure la pantalla principal.

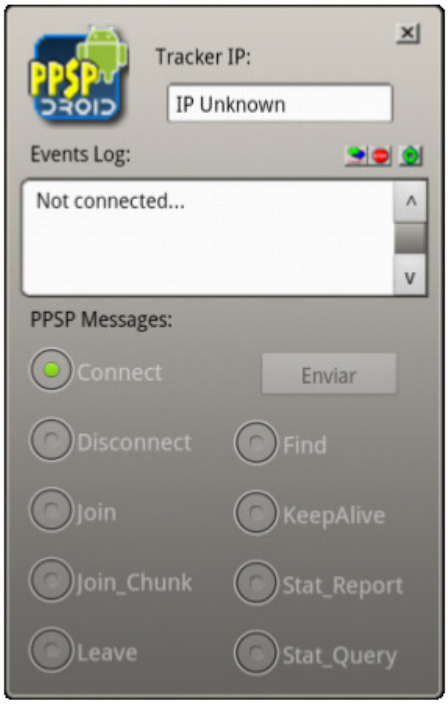

Fig. 7.4.- Pantalla principal del PPSP Droid Node

Un cop vista la pantalla principal, observem que hi ha diverses informacions i botons distribuïts en blocs per tota la interfície. Si ens aturem a veure detalladament aquests diferents blocs que conformen la finestra principal, trobem:

- **Tracker IP:** Com diu el seu nom, ens indica la IP del *Tracker* al qual ens volem connectar. Un cop tocada la casella per introduir la IP, aquesta l'haurem d'indicar a una nova pantalla com la que es veu a continuació:

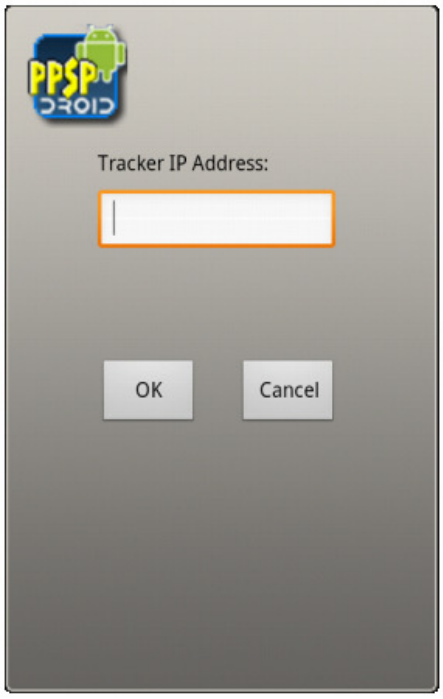

Fig. 7.5.- Pantalla de Tracker IP Address

Es pot introduir una IP que haurà de ser correcta i guardar-la prement *OK*, o sortir de la pantalla amb *Cancel*, tornant en ambdós cassos a la pantalla principal. Novament a la pantalla principal, veiem que just a sobre de la casella de *Tracker IP* hi ha un botó per tancar l'aplicació () a la dreta i el logo del programa a l'esquerra.

- **Events Log:** Situat més al centre de la pantalla principal, en aquest requadre es veurà un log dels esdeveniments que succeeixin a l'aplicació. Té un historial de missatges que mostra fins a cent missatges (un cop entrin més, es van esborrant els més antics). Es podria dir que és un apartat merament informatiu. A més, sobre el requadre dels events, hi ha tres botons que serveixen per connectar i desconnectar amb el Tracker, i per resetejar els events, deixant el requadre buit:  .

- **PPSP Messages:** En aquest apartat trobem un recull de *radio buttons* i un botó normal que serveix per enviar. Amb els diferents *radio buttons* el que fem es triar quin dels tipus de missatge de petició de PPSP volem enviar cap al *Tracker*, i prement el botó 'Enviar', s'enviarà. Cal comentar que aquest apartat no té sentit si no hi una connexió amb un *Tracker* establida. Per aquest motiu, si el node no està connectat amb cap *Tracker*, aquest apartat romandrà deshabilitat fins que es produeixi una connexió satisfactòria.

En tots els dispositius preparats per funcionar amb *Android* existeixen una sèrie de botons al terminal, on els botons *Home* i *Menu* són els més transcendents. Per aquest motiu, per aprofitar les característiques d'*Android*, s'ha implementat un menú amb diverses opcions que es crida un cop es prem em botó de menú del dispositiu.

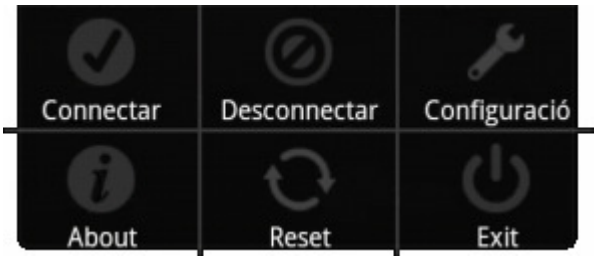

Fig. 7.6.- Menú principal de l'aplicació

- **Connectar:** Com el seu nom indica, amb aquest botó el que es fa és una connexió amb el *Tracker*. Per poder fer això primer hem d'haver introduït una IP correcta (la del *Tracker*) o sinó obtindrem un avís d'error. En cas d'estar ja connectats i tornar a usar aquest botó sense haver desconnectat, el programa també avisarà per informar d'aquest error.

- **Desconnectar:** Contràriament al botó de *Connectar*, aquest botó serveix per desconnectar del *Tracker*. Si el premem abans d'haver connectat exitosament amb un *Tracker*, l'aplicació ens donarà un error per avisar-nos de que no estem connectats i per tant no podem desconnectar.

- **Configuració:** Prement aquest botó, entrarem a la pantalla de configuració, on es poden indicar paràmetres com el *Peer ID* o el nombre de P-Types que volem enviar als missatges d'estadístiques, a un seguit de subpantalles. Aquestes subpantalles de configuració les veurem a continuació, un cop acabat de mirar la resta d'opcions del menú.

- **About:** Aquest botó ens mostra un *popup* amb informació sobre l'aplicació (nom, versió, data, etc.).

- **Reset:** Prement aquest botó el que s'aconsegueix es reiniciar el node i deixar-lo com si el programa acabés d'iniciar-se. Tal i com diu el seu nom, es fa un reset de l'aplicació.

- **Exit:** Botó per sortir de l'aplicació i tancar tot el que pugui estar corrent, així no queda executant-se en *background* ni consumint recursos al dispositiu.

Un cop vist el menú principal de l'aplicació, cal passar a veure les pantalles de configuració tal i com havíem comentat. Un cop es prem el botó de configuració del menú, la pantalla que apareix és la següent:

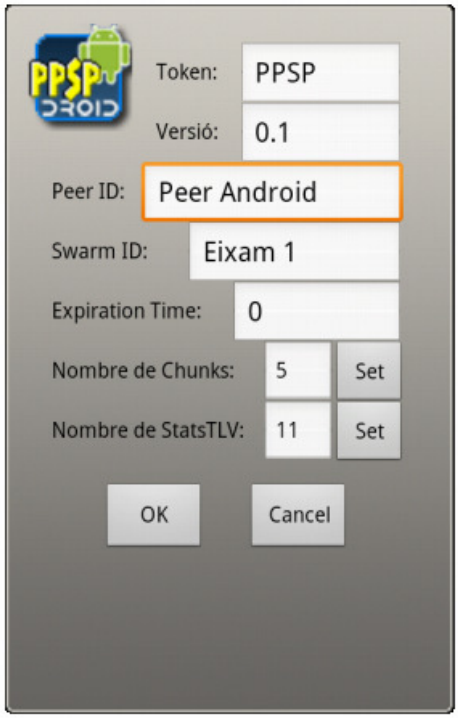

Fig. 7.7.- Pantalla de configuració de l'aplicació

- **Token:** És l'identificador del protocol. Ha de ser PPSP per funcionar bé.

- **Versío:** Per indicar la versió del protocol usada. Actualment s'usa la 0.1.

- **Peer ID:** L'identificador del node. Pot anomenar-se de qualsevol forma.

- **Swarm ID:** És l'eixam al que es vol connectar el *peer*.

- **Expiration Time:** Temps d'expiració pels JOIN que pugui fer el *peer*.

- **Nombre de Chunks:** Prement el botó de Set, obrirem una nova pantalla on podem indicar de quins *chunks* disposem, afegint o esborrant *chunks*.

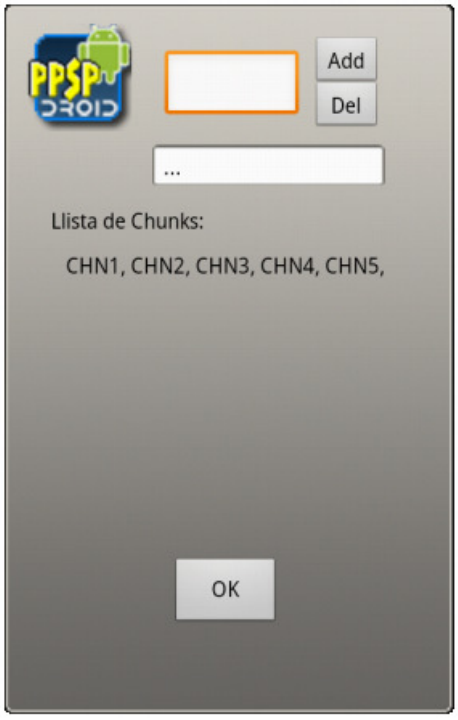

Fig. 7.8.- Pantalla de configuració de chunks

- **Nombre de StatsTLV:** Prement el botó de Set, una nova pantalla s'obrirà i en aquesta podem indicar els diferents P-Types dels que volem informar o rebre informació amb els missatges estadístics.

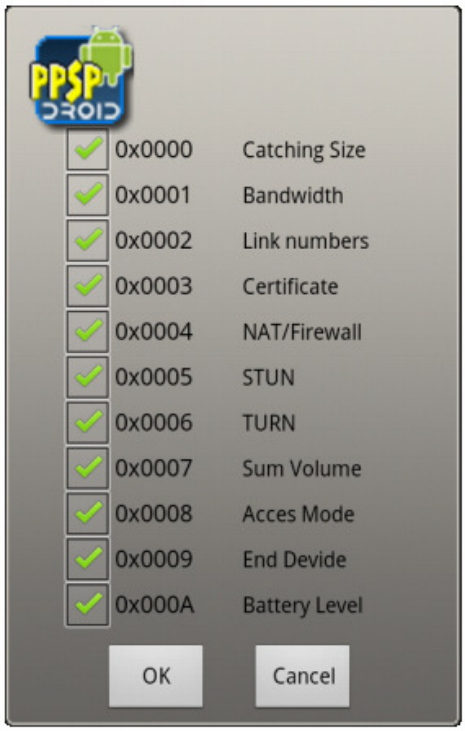

Fig. 7.9.- Pantalla de configuració d'StatTLVs

7.5. Estudi de la sobrecàrrega de la xarxa

Hem de tenir present que PPSP és un protocol que contínuament està enviant informació a la xarxa. Deixant de banda les qüestions de tràfic, que no les contempla el protocol, la informació de senyalització ha de ser enviada i actualitzada constantment per tenir, per exemple, unes llistes de peers el més actualitzades possibles. Missatges com KEEPALIVE o STAT_REPORT, s'envien de forma constant cap a la xarxa, per indicar que el node continua viu i per reportar les estadístiques respectivament. Degut a això, en una xarxa a la que es trobin molts peers, es poden produir situacions de congestió o saturació.

Els temps de reenviament entre un missatge periòdic i un altre no han sigut definits encara pel *Working Group* de PPSP, deixant el tema pendent de discussió per més endavant. Tot i això, és un tema interessant de tractar i d'estudiar.

Per entendre la freqüència amb la que s'haurien d'enviar missatges periòdics com poden ser el missatge PPSP de KEEPALIVE o els missatges d'estadístiques, podem donar un cop d'ull a la definició d'un altre protocol de P2P, el *P2P Chat Protocol*[41].

7.5.1. P2P Chat Protocol

El *P2P Chat Protocol* és un protocol basat en una arquitectura de chat per *Mobile Ad Hoc Networks* (MANETs). Això significa que tots els nodes de la xarxa són iguals des del punt de vista del protocol. Per establir la xarxa dels participants del chat els nodes només han de descobrir nodes potencials al seu voltant (veïns). Degut a que els nodes mòbils deixen i entren a la xarxa en qualsevol moment, la topologia de xarxa segurament canviarà contínuament i xarxes particionades emergiran. Per aquesta raó, l'arquitectura de chat ha de ser tolerant per poder gestionar aquests esdeveniments.

Aquest protocol té una sèrie de tipus de missatges que intercanviaran els *peers*. Ja que no és el tema que ens ocupa, ens centrarem només en comentar el missatge HELLO, deixant de banda la resta de missatges (ACK, NACK, CHANNEL, etc.).

El missatge HELLO és un tipus de missatge que s'envia dins del *P2P Chat Protocol* de forma periòdica per cada node. L'interval entre dos missatges HELLO satisfactoris ha de ser de 2 segons amb un marge de ±0.5 segons. És a dir, un *peer* ha de triar un nou interval de forma aleatòria entre 1.5 i 2.5 segons per cada missatge enviat de forma individual per tal de reduir la probabilitat de col·lisió. El missatge HELLO s'usa per detectar altres *peers* a la xarxa local. Si no es rep cap missatge de tipus HELLO durant tres intervals successius, el node remot s'eliminarà del rang de transmissió.

7.5.2. Rendiments estimats d'una xarxa streaming P2P

Tal com s'ha vist a l'apartat anterior, un temps de 2 segons seria suficient per anunciar-se a una xarxa P2P. Tot i això, hem de tenir present que els canvis que es poden produir en un node, segurament no seran tan freqüents i, per tant, aquest temps és més petit del necessari realment. Es necessitarà, per tant, arribar a un compromís amb el nombre de *peers* a la xarxa.

Es pot fer un càlcul estimat i aproximat dels requeriments del mètode basat en *Tracker* per què un *peer* pugui descobrir contingut específic a la xarxa. El mètode basat en Tracker és el mètode en el que està basat PPSP actualment, ja que té més avantatges que el mètode basat en DHT. En general, el mètode basat en *Tracker* és molt més ràpid que el mètode basat en DHT, amb menor càrrega de tràfic a la xarxa, tot i que el mètode basat en DHT necessita menys requeriments al host. Aquest últim fet, es pot solucionar usant múltiples *Trackers*. Una forma de fer això és utilitzant el protocol RELOAD: *PPSP Tracker Usage for Reload*[42].

Per fer els càlculs, assumirem que hi ha D recursos compartits per N *peers* a un sistema *peer to peer*. Per tant, per *P2P streaming*, podríem definir:

- **N:** nombre d'usuaris actius a una xarxa *streaming* P2P, que pot ser de 10 milions; i **D:** nombre de canals (*live streaming*) o vídeos (VoD), sobre 100 mil en total.

Un cop definit aquests paràmetres, es poden estudiar els rendiments de la recerca d'eficiència (*lookup efficiency*), el tràfic de la xarxa (*network tràfic*) i els requeriments de host (*host requirements*) tant pel descobriment de recursos (*Resource Discovery*) i la detecció de *chunks* (*Chunk Discovery*):

- **Resource Discovery:** els nivells secundaris només comparen el rendiment de descobriment de la informació dels recursos.
    - **Lookup Efficiency:** definim el següent paràmetre:

      - RTT: *Round-Trip delay Time*, és el temps que tarda a retornar un paquet des d'un emissor al mateix havent passat pel destí. RTT mig a la xarxa = 200 ms.

Missatges de Lookup	O(1)
Operacions de Lookup	O(1)
Latència de Lookup	O(1)*RTT = 200ms

      La latència és relativament baixa i no suposa un problema greu. La notació O(1) [43] és una funció extremadament propera a 1 per valors d'entrada molt grans.

- **Network Traffic:** Podem definir els següents paràmetres:

  - T: cada *peer* demana nous recursos cada T segons, on T = 60s

  - S: mida mitja d'un missatge de petició/resposta, S = 1KB

Nombre de missatges per segon	(N/T)*2 = 3,33*100.000
Mida de missatges per segon	(N/T)*2*S = 0,33 GB
Nombre de missatges per Join/Leave	O(1)

  És una gran càrrega de trànsit per la xarxa, però prou assumible.

- **Host Requirement:** Podem definir els següents paràmetres:

  - T: cada peer demana nous recursos cada T segons, on T = 60s

  - S: mida mitja d'un missatge de petició/resposta, S = 1KB

  - C: un *peer* te C recursos, en aquest cas poderm posar C = 10

  - P: cada *peer* està representat per P Bytes, on P = 20B

Requeriment de memòria	N*C*P = 2GB
Nombre de peticions rebudes per segon	N/T = 1,67*100.000
Mida dels missatges de petició/resposta per segon	(N/T)*2*S = 0,33 GB

  La càrrega de missatges que ha de soportar el *Tracker* en aquestes condicions és elevada. Es pot considerar l'ús de múltiples *Trackers* per pujar el rendiment.

- **Chunk Discovery:** el nivell principal també compara el rendiment de descobriment de la informació dels *chunks*.

  - **Lookup Efficiency:** definim el següent paràmetre:

    - RTT: *Round-Trip delay Time*, o temps que tarda a retornar un paquet des d'un emissor al mateix havent passat pel destí. RTT mig a la xarxa = 200 ms.

    - M: cada *peer* es comunica amb M peers, on M = 20

	Tracker Side	**Peer Side**
Missatges de Lookup	O(1)	M*O(1) = 20
Operacions de Lookup	O(1)	O(1)
Latència de Lookup	O(1)*RTT = 200ms	O(1)*RTT = 200ms

    La latència és baixa a les bandes de *Tracker* i de *Peer*. No suposa un problema.

- **Network Traffic:** Podem definir els següents paràmetres:

  - T: cada *peer* demana nous recursos cada T segons, on T = 60s

  - S: mida mitja d'un missatge de petició/resposta, S = 1KB

  - I: cada *peer* envia informació (*gossip messages*) cada I segons, I = 10s

  - R: *rate* del vídeo, el definim com R = 32KB/s

  - Z: mida d'un *chunk*, el definim com Z = 16KB

	Tracker Side	**Peer Side**
Nombre de missatges per segon	(N/T)*2 = 3,33*100.000	M*(N/I)*2 = 4*10.000.000
Mida de missatges per segon	(N/T)*2*S = 0,33 GB	M*(N/I)*2*S = 40GB

  És una gran càrrega de trànsit per la xarxa, però prou assumible.

- **Host Requirement:** Podem definir els següents paràmetres:

  - T: cada peer demana nous recursos cada T segons, on T = 60s

  - S: mida mitja d'un missatge de petició/resposta, S = 1KB

  - C: un peer te C recursos, en aquest cas poderm posar C = 10

  - P: cada peer està representat per P Bytes, on P = 20B

  - Bm: mida del *bitmap*, Bm = 1KB

  - I: cada *peer* envia informació (*gossip messages*) cada I segons, I = 10s

	Tracker Side	**Peer Side**
Requeriment de memòria	N*C*P = 2GB	M*Bm = 20KB
Nombre de peticions rebudes per segon	N/T = 1,67*100.000	M/I = 2
Mida dels missatges de petició/resposta per segon	(N/T)*2*S = 0,33 GB	(M/I)*2*S = 4KB

  La càrrega de missatges que ha de soportar el *Tracker* en aquestes condicions pel *chunk discovery* és elevada. En aquest cas també es pot considerar l'ús de múltiples *Trackers* per pujar el rendiment.

Amb aquests càlculs hem aclarit la càrrega que han de suportar *Tracker* i xarxa a un *streaming* P2P amb elevats usuaris i recursos però, encara no s'ha determinat quina podria ser la cadència d'enviament dels missatges periòdics. Abans, es comentarà de forma breu l'ús de múltiples *Trackers* amb RELOAD: *PPSP Tracker Usage for RELOAD*.

7.5.3. PPSP Tracker Usage for RELOAD

PPSP assumeix que un *Tracker* centralitzat es fa servir per comunicar amb els *peers* de PPSP per localització i registre de contingut. L'índex de contingut està emmagatzemat al *Tracker* amb informació de localització sobre quins *peers* tenen el contingut.

Tot i això, el *Tracker* centralitzat de forma lògica pot ser realitzat també mitjançant un clúster de *Trackers* distribuïts geogràficament o desplegats en múltiples servidors dins d'un centre de dades (o *data center*), de forma que així es pot incrementar la disponibilitat de contingut, la robustessa del servei i la fiabilitat i escalabilitat de la xarxa. La gestió i la localització de l'informació d'indexat són comportaments totalment inters del clúster de *Trackers*, que serà invisible pels peers de PPSP. La senyalització entre els *peers* i els *Trackers* seguirà sent el simple mecanisme de petició-resposta definit al *PPSP Tracker Protocol*. Per tant, els nodes *Tracker* per si mateixos construeixen i manetenen una xarxa P2P superposada, tal i com es pot veure a la figura següent.

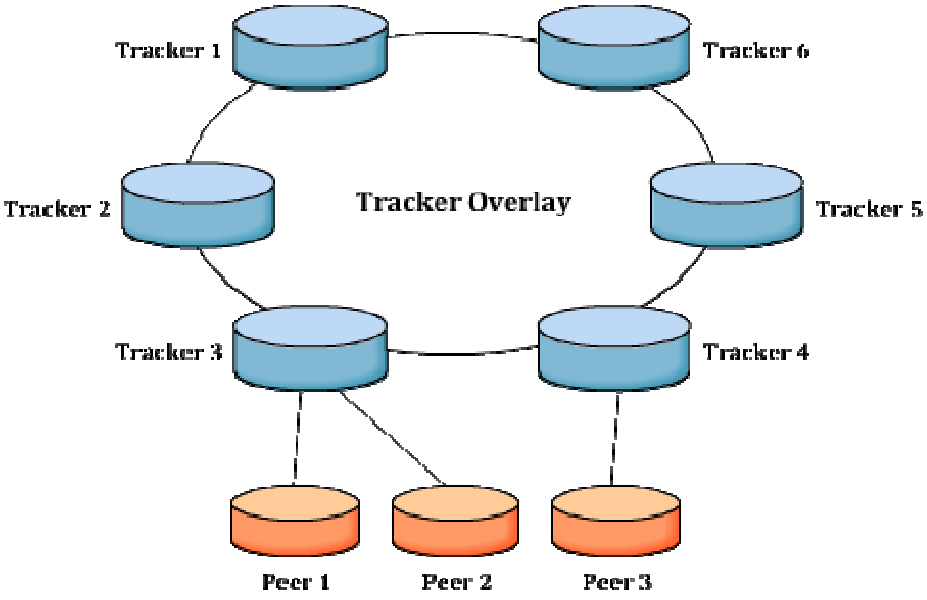

Fig. 7.10.- Peers connectats amb una xarxa Tracker Overlay amb el Tracker Protocol

Hi ha quatre funcions bàsiques a una xarxa superposada de *Trackers*:

- **Registre del contingut i el canal d'informació d'indexat:** els *peers* registren/actualitzen els seus continguts i canals a un *Tracker* de connexió. Aquest *Tracker* té l'avantatge de la funcionalitat d'emmagatzemament de dades de RELOAD per emmagatzemar l'informació d'indexat als nodes *Tracker* de la xarxa superposada de *Trackers* en conseqüència.

- **Buscar l'índex d'un contingut/canal:** Un cop un *peer* de PPSP busca per cert contingut/canal, fa la petició cap a una connexió local de *Tracker* tal i com està especificat al *PPSP Tracker Protocol*. Si les dades no poden ser trobades al *Tracker* de forma local, el *Tracker* localitzarà l'informació d'indexat requerida al *Tracker Overlay* en nom del *peer* que fa la petició. Un cop la llista de *peers* s'obtingui, el *peer* establirà comunicacions amb els *peers* de la llista de *peers* tal i com es defineix al *PPSP Peer Protocol*.

- **Registre de l'estat dels peers,** que el registren/actualitzen al *Tracker Overlay*.

- **Recerca de l'estat d'un cert peer,** que el *Tracker Overlay* podria buscar.

7.5.4. Model matemàtic

Basant-nos en els càlculs i estimacions fetes anteriorment, i usant les característiques d'altres protocols de P2P que usen missatges d'anunciament periòdics, podem trobar una cadència adequada per l'enviament de missatges periòdics amb PPSP.

Mesurem la fiabilitat per un senzill criteri: la probabilitat de l'èxit de recerca (*lookup success*). Aquesta probabilitat depèn de dos factors: Si els nodes Tracker requerits per respondre la petició de recerca estan corrent i si aquests mateixos nodes estan sobrecarregats. El primer factor es pot avaluar basant-nos amb la teoria de fiabilitat estàndard. D'altra banda, el segon factor es pot avaluar amb la simple teoria de cues.

7.5.4.1. Fiabilitat

Assumim que els nodes *Tracker* tenen infinita capacitat (el rendiment no és problema).

Per la fiabilitat, el disseny del *Tracker* es pot modelar tant com un conjunt d'unitats en paral·lel or un grup d'unitats en sèrie on cada unitat pot estar composada per sets d'unitats en paral·lel, com es mostra a la figura següent. Viem el disseny d'un *Tracker* basat en servidor configurat de forma manual amb cert grau de replicació.

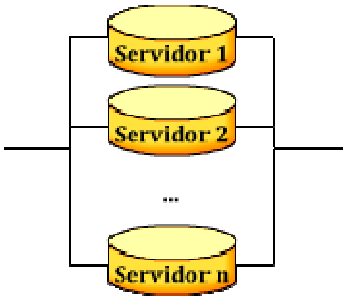

Fig. 7.11.- Model de fiabilitat d'un Tracker basat en servidors

Fem que R_{server} denoti la fiabilitat d'aquest disseny de *Tracker*, i definim R_s com la fiabilitat de servidors desplegats, llavors:

$$R_{server} = 1 - (1 - R_s)^n$$

on *n* és el nombre de nodes desplegats que actuen com el *Tracker* per un objecte.

El terme R_s pot ser expressat també en termes de *mean time to failure* i *mean time to repair* per dos tipus de nodes *Tracker*; a més, diferents nodes *Tracker* poden tenir diferents valors de fiabilitat.

7.5.4.2. El factor de rendiment

En realitat, tant els servidors com els peers desplegats tenen capacitat finita, i per tant es poden veure sobresaturats sota una càrrega elevada d'informació. Per tant, la fiabilitat del node R_s es pot veure com el producte dels següents factors:

$$R_s = P_{up} \cdot P_{queued} \cdot P_{served}$$

El primer dels termes, P_{up}, pren el valor original de R_s. L'últim terme, P_{served}, és la probabilitat de que el node *Tracker* es mantingui fins que la petició es serveixi i és una probabilitat bastant propera a 1, i es pot ignorar en molts cassos. El segon terme, P_{queued}, és la probabilitat de que el node *Tracker* no estigui sobrecarregat, i pot derivar d'un simple model de cues com pot ser M/M/1/h, on *h* és la longitud de la cua (el nombre de peticions de recerca simultànies acceptades pel node *Tracker*). Els *rates* de petició de servei, μ_s, són una propietat dels nodes *Tracker*. El *rate* d'arribada de petició, λ, ve donat per la càrrega de treball, derivat de la taxa d'arribada dels *peers* i el nombre mig d'objectes als que cada *peer* accedeix mentre està al sistema. Basat en aquests paràmetres, podem obtenir la fiabilitat d'un *Tracker* basat en servidor tal com:

$$R_s = P_s \cdot \left(1 - \frac{\rho_s^h - \rho_s^{h+1}}{1 - \rho_s^{h+1}}\right)$$

$$\text{on, } \rho_s = \frac{\lambda}{n \cdot \mu_s}$$

assumint que les peticions estan distribuïdes equitativament en els *n* nodes servidors.

Finalment, Rs pot ser connectat a la fórmula de fiabilitat per obtenir la tassa d'èxit de les operacions de recerca.

7.5.4.3. Representació gràfica del model

El model de *Tracker* pot ser usat per avaluar alternatives de disseny de *Tracker* per diverses xarxes i paràmetres de càrrega de treball. També es pot acurar més per a servir com eines de planificació de capacitat per desplegar les funcions de *Tracker* per obtenir els objectius de fiabilitat i rendiment. A partir del que s'ha vist als sub-apartats anteriors amb la fiabilitat i el factor de rendiment, es poden fer certes observacions:

La fiabilitat d'un *Tracker* basat en servidor es pot deteriorar ràpidament sota una gran càrrega de tràfic, baixant la probabilitat d'èxit de recerca si el nombre de *peers* augmenta. Aquest problema és solucionable amb planificació de capacitat acurada.

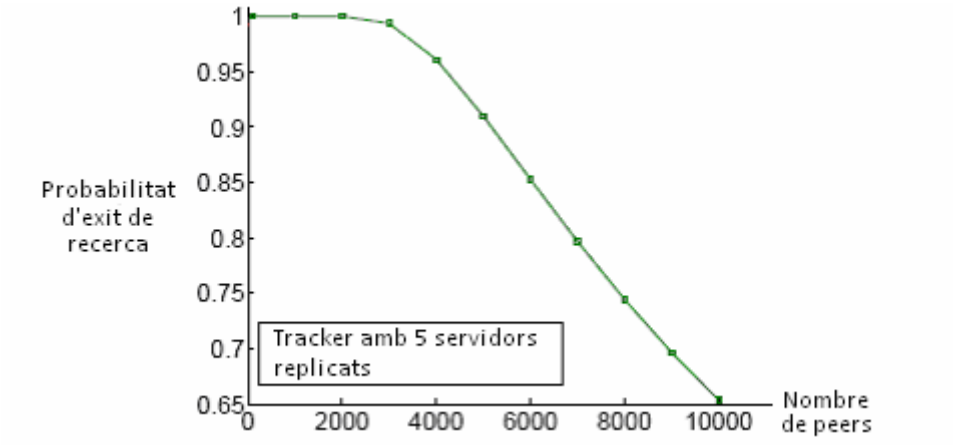

Fig. 7.12.- Probabilitat de l'éxit de recerca amb influència del nombre de peers

L'increment del paràmetre *mean time to failure* del node *Tracker* pot millorar la fiabilitat del *Tracker*, tal i com s'esperava. Però és interessant veure que la sensibilitat d'aquest paràmetre és alta, degut a que aquest model assumeix que un servidor necessita un temps de reparació aleatori abans de tornar-se disponible un altre cop.

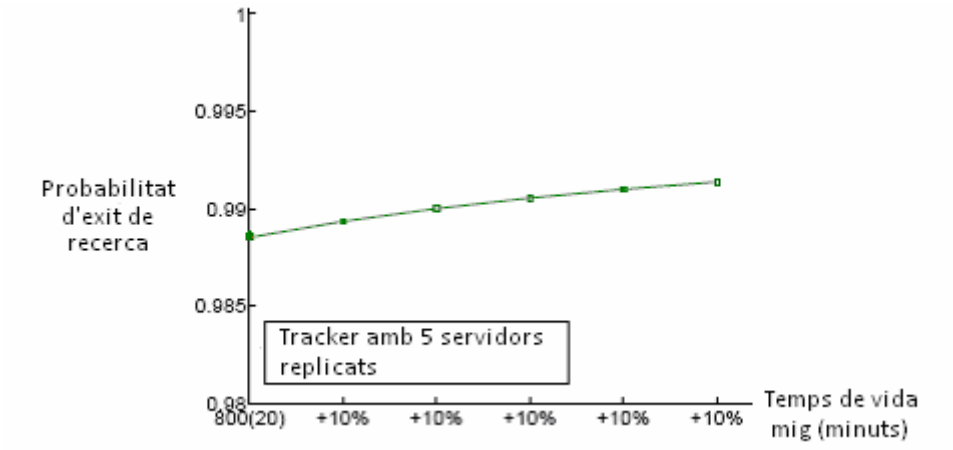

Fig. 7.13.- Probabilitat de l'éxit de recerca amb influència del temps de vida mig

L'increment de la replicació de *Trackers* millora la fiabilitat. De fet, a les anteriors gràfiques s'ha usat una replicació de 5 servidors, ja que veiem que la probabilitat de recerca augmenta considerablement a partir d'aquest cas.

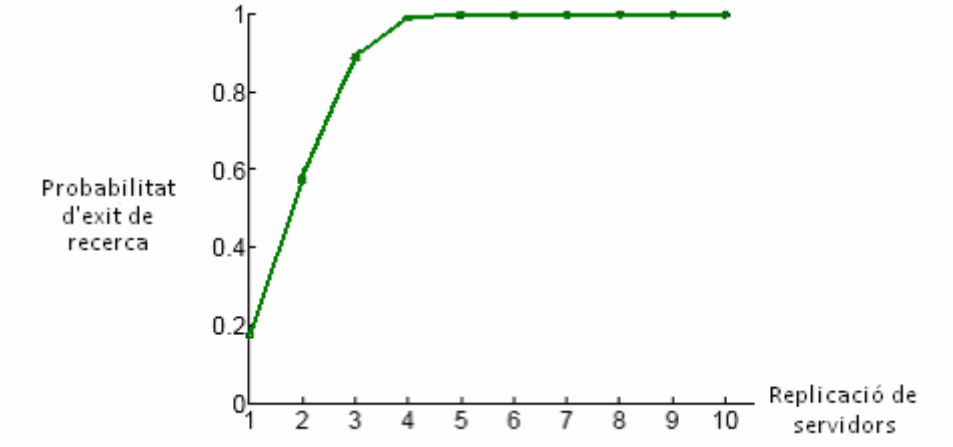

Fig. 7.14.- Probabilitat de l'éxit de recerca amb influència de la replicació

Més enllà de la fiabilitat, existeixen altres dimensions interessants per quantificar el rendiment del *Tracker*, com per exemple la càrrega del *Tracker* per diferents taxes d'accés i els rendiments de cost de localitzar contingut.

7.5.5. Conclusions

Arribats a aquest punt, hem vist que la càrrega de tràfic i de missatges que ha de suportar un Tracker és elevada, i que això pot influir negativament a la seva fiabilitat si el sobresaturem. Es per això que s'ha de trobar un compromís acceptable alhora d'enviar els missatges periòdics. És a dir, si el temps entre missatges és massa baix, s'enviaran més missatges dels necessaris i es contribuirà a crear un coll d'ampolla que saturarà innecessariament la xarxa. D'altra banda, si és massa baix, les llistes de *peers* i altres llistes no estaran prou actualitzades, a part de que els *peers* no seran conscients de forma immediata dels *peers* que hagin desconnectat.

Per veure de forma pràctica això, s'ha afegit un petit mòdul a l'aplicació desenvolupada en C++, per bombardejar amb missatges el *Tracker*, de forma que es pugui veure el temps que triga a respondre. Tot i que un *Tracker* ha d'estar preparat per assumir una càrrega de tràfic molt elevada, per limitacions, el Tracker programat el testejarem amb un màxim de mil missatges enviats alhora. Primer observarem el temps de resposta del *Tracker* si enviem un conjunt de mil missatges seguits durant un segon. És a dir, és com si el temps dels missatges periòdics fos d'un segon entre missatges, i per tant tots es concentren al mateix interval de temps, de forma que es pot veure un excés de retard.

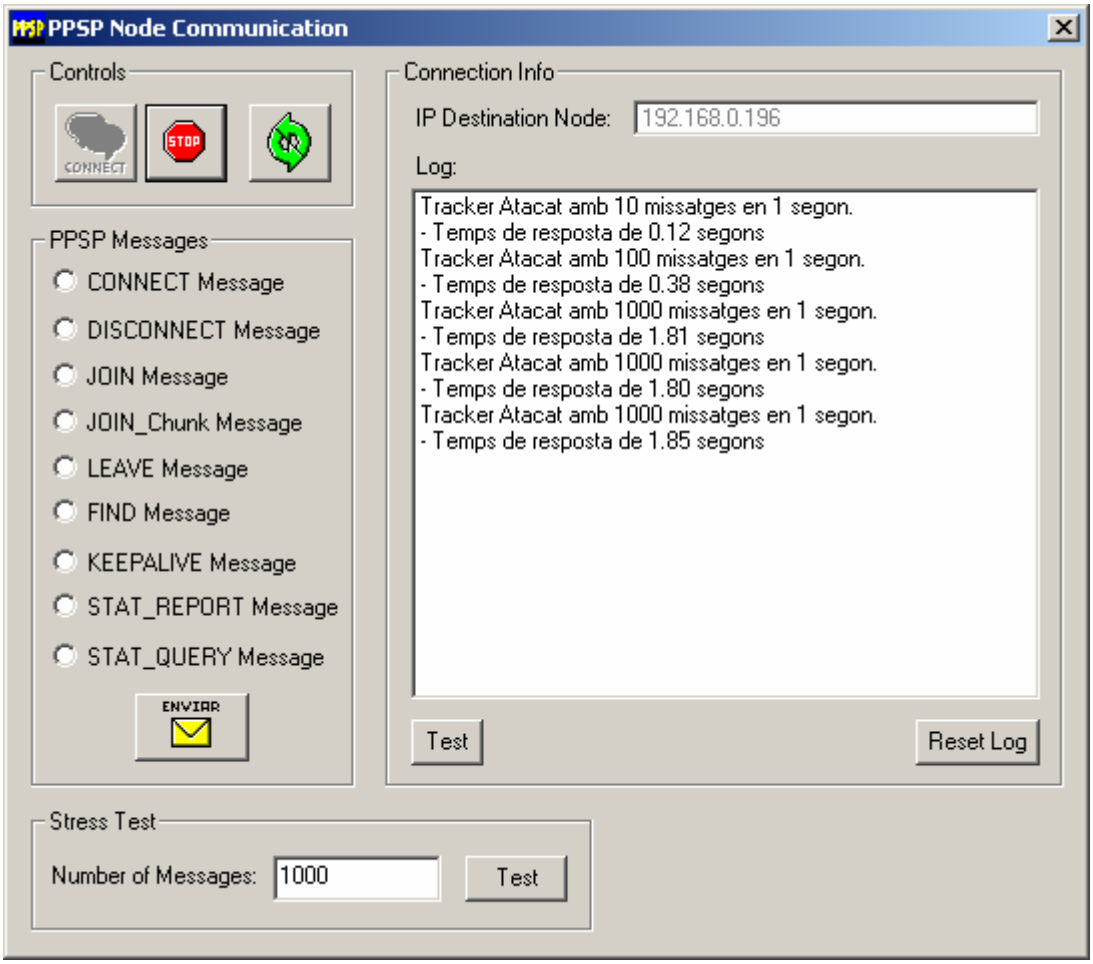

Fig. 7.15.- Temps de resposta per mil missatges

El temps de resposta és el temps que tarda el *peer* en rebre l'última resposta des de que ha enviat la primera petició. Observem que la mitja és 1,82 segons, i sabem que el temps total per enviar mil missatges ha estat d'un segon, per tant el temps de resposta mig real ha sigut de 0,82 segons. Si en comptes de mil missatges fossin 10, el temps de resposta serien 0,12 segons, un temps imperceptible comparat amb l'1,82.

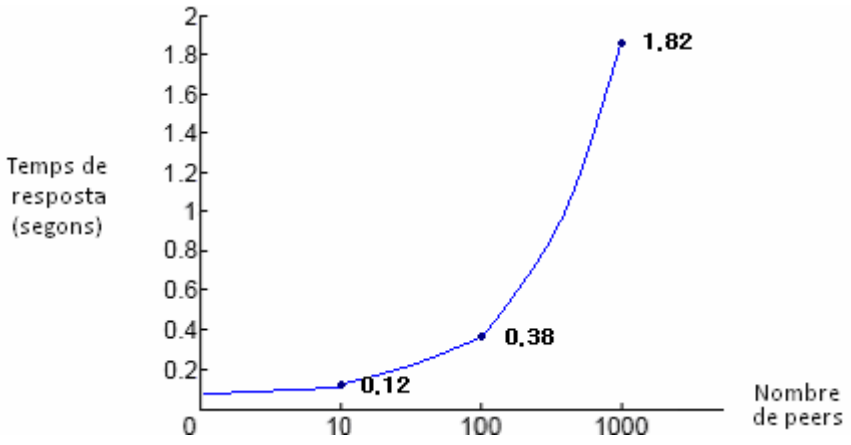

Fig. 7.16.- Missatges enviats durant un segon

Si per contra, féssim que el temps entre missatges fos de 30 o 60 segons, en comptes de 1 o 2, com hem vist que es fa al *P2P Chat Protocol*, la probabilitat de col·lisions entre missatges disminueix, ja que no estaran concentrats tots al mateix interval de temps reduït, i estarien més distribuïts en el temps, de forma que el *Tracker* va molt més lleuger per respondre al no tenir tanta càrrega de missatges. De fet, fent una càlcul ràpid, es podria dir que la distribució de la càrrega de missatges, si fos equitativa, seria d'uns 33 o 34 missatges per segon què, comparats amb els mil missatges per segon anteriors, clarament es veu una reducció de la càrrega.

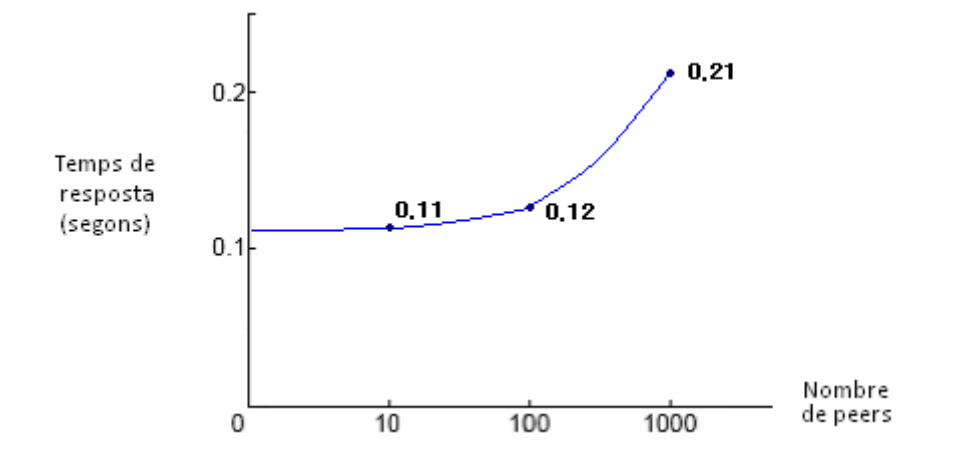

Fig. 7.17.- Missatges enviats durant trenta segons

Amb tot això, i les dades obtingudes anteriorment a la resta de sub-apartats, arribem a la conclusió de que una bona opció seria la d'optar per un temps de 30 o 60 segons entre cada missatge periòdic, per tal de no saturar la xarxa i evitar que la informació de les llistes quedi massa temps desactualitzada. Llavors, des del *Tracker*, s'hauria de tenir present aquesta consideració de temps i, en cas de no rebre tres missatges consecutius (igual que passa amb els missatges HELLO del *P2P Chat Protocol* o inclús amb els missatges d'anunci de sessió del *Session Anouncement Protocol*[44]), en el cas dels missatges de KEEPALIVE, el *Tracker* assumirà que el *peer* ha desconnectat.

7.6. Resultats obtinguts

En aquest apartat es dedicaran unes línies a donar-li un cop d'ull als resultats obtinguts pel programa durant les diverses, i exhaustives, proves que s'han fet. Per acompanyar les diferents explicacions, s'inclouen captures de pantalla tant de l'aplicació desenvolupada per *Android*, com de l'anterior aplicació en C++ (que servirà de fer de *Tracker* i de *peers* en nodes diferents) i com de *Wireshark*, amb el *protocol dissector* de PPSP programat en Lua; amb els diferents esdeveniments i cassos que es tracten. D'aquesta forma l'explicació és molt més visual i aclaridora.

Abans de començar, s'han de preparar les diferents aplicacions. Cal dir que per tal de que l'aplicació programada en C++ funcioni com a *Release* en qualsevol ordinador que la vulgui arrancar, calen uns fitxers de configuració inicial amb el nom i el format correcte. De no ser així, el programa agafarà uns valors per defecte, minimitzant les opcions i les possibilitats amb les que treballar. Per obtenir més informació sobre els fitxers de configuració de l'aplicació en C++ (*PPSP Node Communication*), consultar la memòria corresponent (*Estudi i implementació del nou protocol de Streaming Peer to Peer, PPSP*) a l'apartat *7.2.4. Fitxers de configuració*. Aquests són els següents:

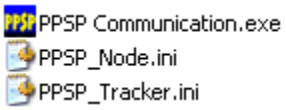

Fig. 7.18.- Fitxers necessaris per l'execució de l'aplicació en C++

A més, cal comentar que degut al que hem vist a l'apartat de compatibilitat entre Java i C++, s'han hagut de fer unes petites modificacions a la forma d'enviar les dades dels missatges a través dels *sockets* de l'aplicació *PPSP Node Communication* per tal de fer-la compatible amb la nova aplicació, a la vegada més així les dades s'envien en un format estàndard compatible amb varis llenguatges de programació.

D'aquesta forma, ara el *PPSP Node Communication* pot continuar funcionar igualment com a *peer* o com a *Tracker* i, funcionant com a *Tracker*, pot tenir connectat diversos *peers* que poden ser tant nodes fixes que funcionin amb la mateixa aplicació en mode *peer*, o nodes que funcionin sota el sistema operatiu *Android* amb el *PPSP Droid Node*, que generalment seran dispositius mòbils.

Així doncs, el *PPSP Node Communication*, es troba ja a la seva versió 4.1 que és compatible totalment amb les versions anteriors del programa i, a més, amb el *PPSP Droid Node*.

D'altra banda, tenim l'aplicació desenvolupada per *Android*, que com ja s'ha comentat s'anomena *PPSP Droid Node*. Aquesta està compilada en un únic fitxer amb l'extensió *.apk*, preparat per ser instal·lat a qualsevol terminal que funcioni amb *Android*.

A continuació, podem observar com es veu l'aplicació instal·lada a l'emulador:

Fig. 7.19.- Captura de l'emulador d'Android on es pot veure l'icona del PPSP Droid Node

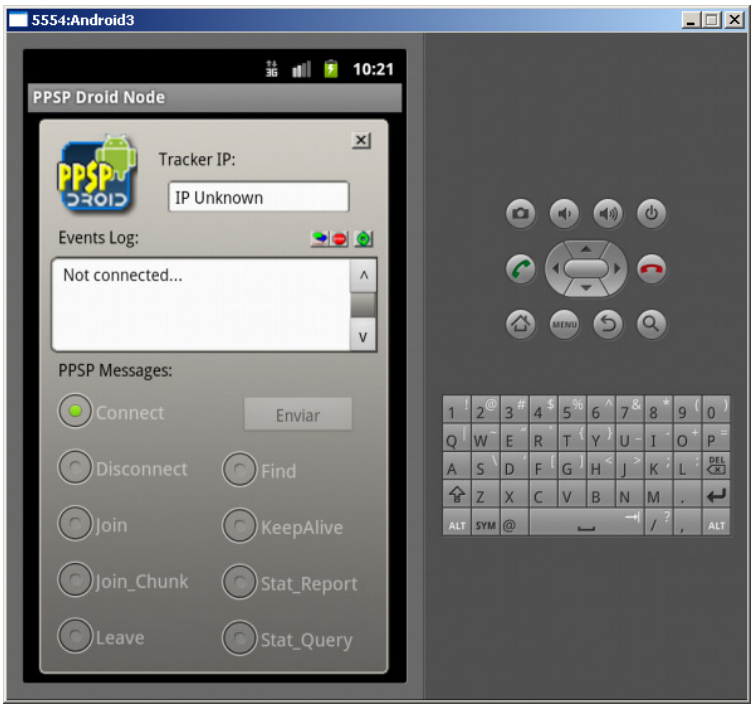

Fig. 7.20.- Captura de l'emulador d'Android on es pot veure el PPSP Droid Node

Tot i que les captures del terminal *Android* es faran amb l'emulador, l'escenari pràctic s'ha implementat també sobre un terminal real. El fet que les captures s'hagin fet amb l'emulador és simplement perquè és molt més nítid per veure-les.

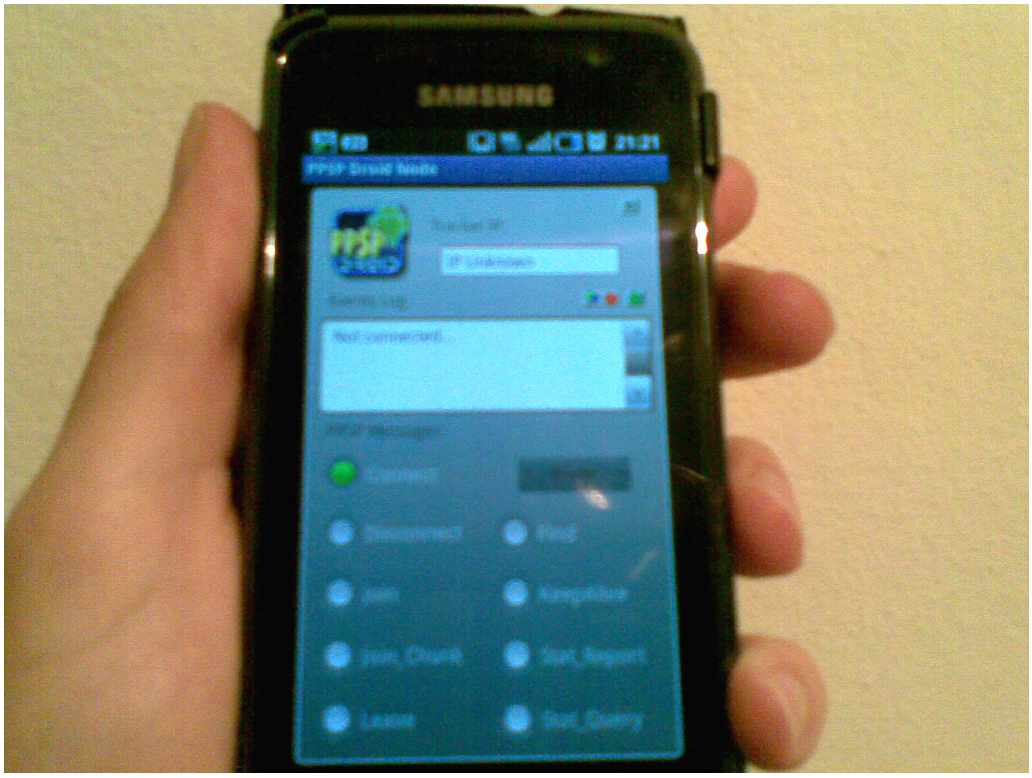

Fig. 7.21.- Terminal Android amb PPSP Droid Node funcionant

L'escenari sobre el qual s'ha treballat per veure resultats consta de quatre nodes, un que fa de *Tracker* i tres que actuen com a *peers*, sent un d'ells un node mòbil funcionant sota *Android*. Tots ells tenen unes IP conegudes i estan a la mateixa xarxa.

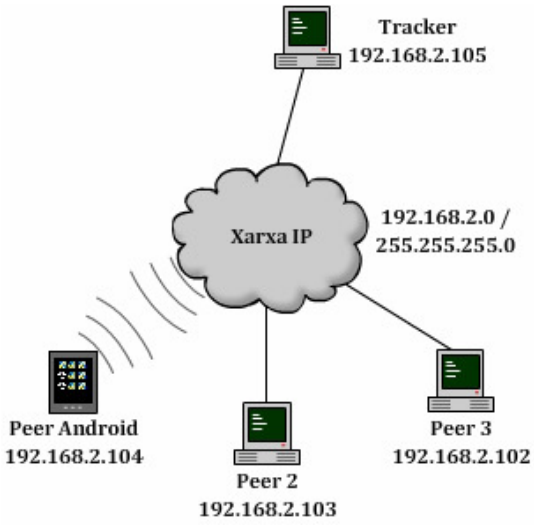

Fig. 7.22.- Escenari pràctic implementat

Vist l'escenari, es pot passar a l'execució del mateix. Anant per passos, mostrem primer la finestra de l'aplicació tal i com es veu al node que fa de *Tracker*. La configuració inicial d'aquest, que indiquem nosaltres al fitxer de configuració inicial, és la que indica que el node treballa com a *Tracker*. Habilitem les funcions estadístiques i d'entrada no posem a *true* cap de les característiques del Tracker que ens serveixen per indicar estats concrets (*Message Not Supported*, *Tracker Overloaded*, etc.), de forma que el Tracker sempre contestarà un missatge de *Succesful* a qualsevol petició, a menys que el missatge no tingui la versió corresponent o no el reconegui com a missatge de PPSP.

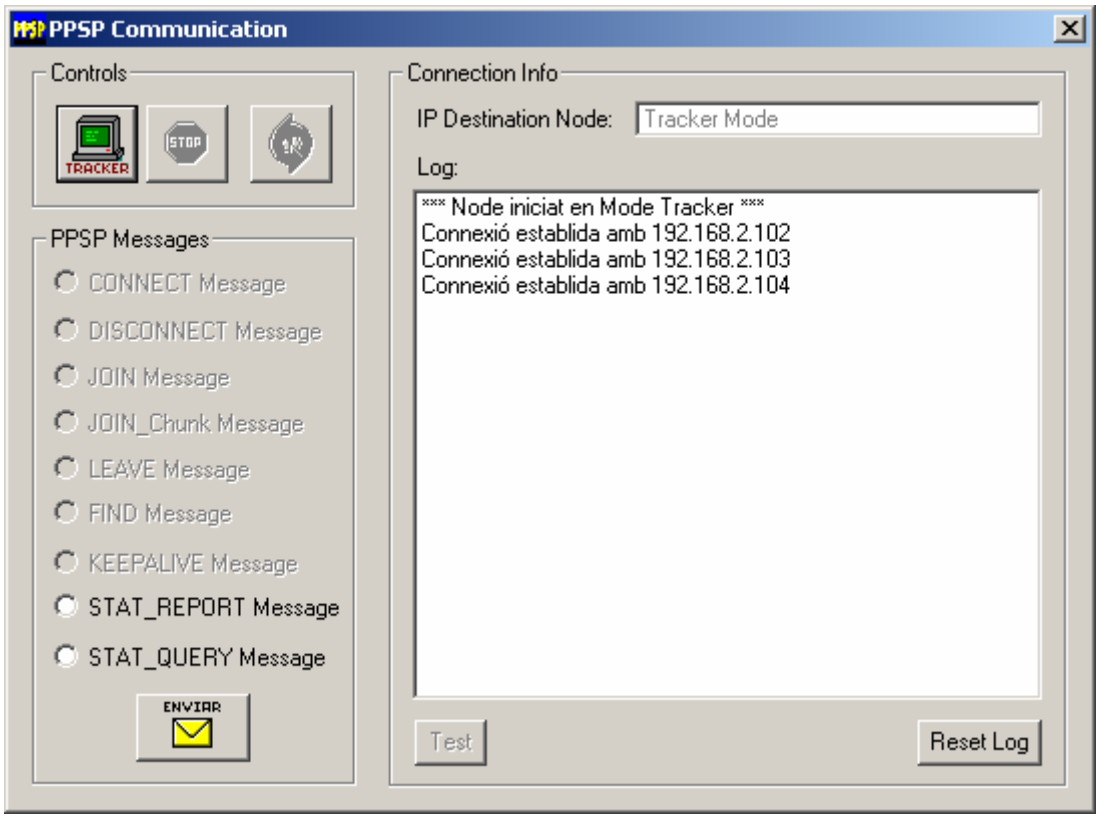

Fig. 7.23.- L'aplicació PPSP Node Communication funcionant com a Tracker

D'altra banda, a cada un dels *peers*, s'ha d'iniciar l'aplicació corresponent indicant, en el cas del *PPSP Node Communication*, a la configuració inicial que no ha d'actuar en mode *Tracker*. El node actuarà com a *peer*. Al *PPSP Droid Node* no cal indicar res, ja que l'aplicació només pot actuar com a *peer*. D'entrada els *peers* tenen els missatges de PPSP deshabilitats, ja que primer s'ha d'establir una connexió amb el node *Tracker*.

Una vegada tenim els nodes iniciats als diferents dispositius, és hora de fer les connexions dels *peers* amb el *Tracker*. Des de cada *peer* basta amb prémer el botó de *Connect* que hi ha a l'apartat *Controls*. Fent això, apareixerà una petita finestra on s'ha d'introduir l'adreça IP del node on hi ha el *Tracker*, en aquest cas 192.168.2.105.

Fig. 7.24.- Connexió d'un peer PPSP Node Communication

Un cop introduïda la IP del *Tracker*, prement *Aceptar*, el *peer* connectarà amb aquest. Amb el *PPSP Droid Node* ja hem vist a l'apartat d'interfície gràfica la subpantalla on es pot configurar la IP del *Tracker*. Aquesta IP serà la mateixa que en el cas anterior.

En aquest punt, els nodes estan connectats al *Tracker* i es pot començar l'intercanvi de missatges del protocol. El *Tracker* pot fer peticions estadístiques i els *peers* poden fer qualsevol petició. Cal dir que les diferents comandes haurien de tenir un cert ordre lògic (no té sentit fer DISCONNECT si no s'ha fet CONNECT). No obstant, el que es pretén es veure l'intercanvi de missatges del protocol després d'haver fet l'estudi del mateix, i així veure l'interactuació i el funcionament. Per tant, l'aplicació no contempla aquest l'ordre dels missatges, de forma que, tot i influir a la base de dades del *Tracker*, es podria dir que són missatges independents entre ells. No obstant, per què l'escenari muntat tingui coherència, les seqüències mostrades seguiran un cert ordre.

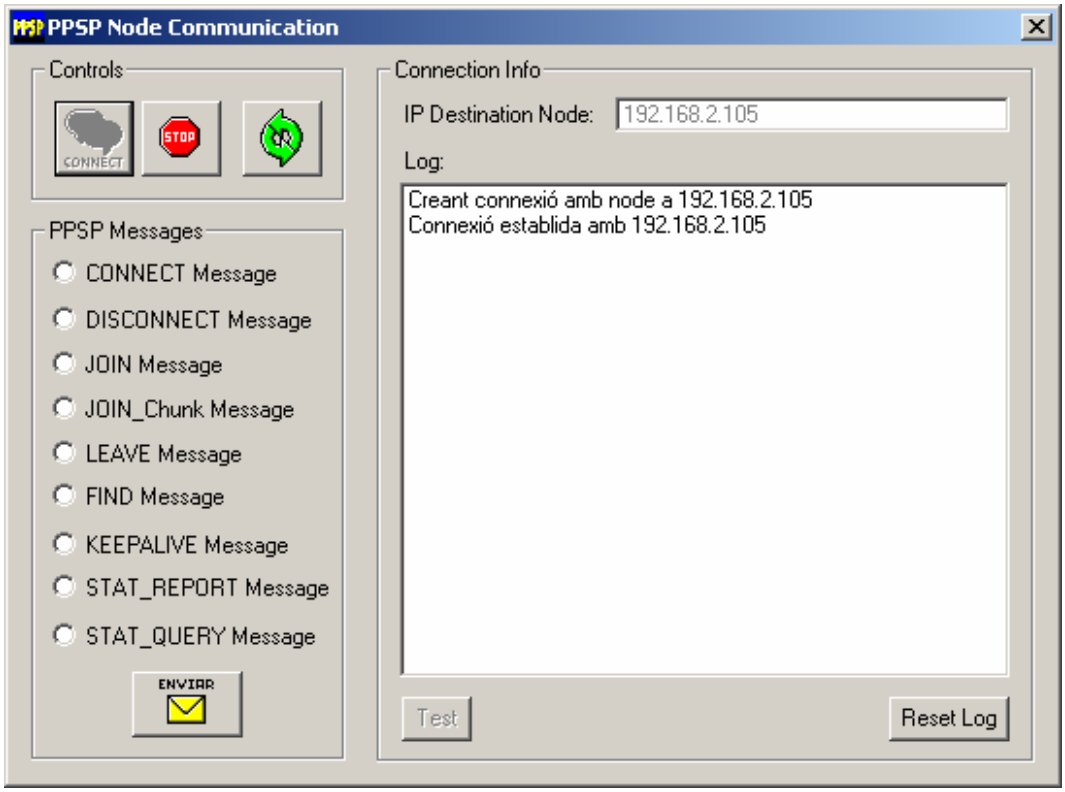

Fig. 7.25.- PPSP Node Communication funcionant com a peer

A la configuració inicial del *Tracker*, hem indicat que hi ha tres eixams amb identificadors d'eixam: *Eixam 1*, *Eixam 2* y *Eixam 3*. Els tres eixams estan buits d'entrada, ja que hem deixat buida la base de dades virtual que es pot configurar. Els *peers* han de fer un missatge CONNECT per establir la connexió amb el *Tracker* i fer un JOIN (o JOIN_CHUNK) per entrar a formar part d'un eixam concret.

El "problema" amb el que ens trobem és que tal i com estan programades ambdues aplicacions, als respectius *logs* només es llisten les seqüències de missatges. Aquí es on entra la funcionalitat de *Wireshark* per capturar trames a la xarxa i el *protocol dissector* de PPSP programat amb Lua per *Wireshark*. És a dir, els missatges que circulen entre els nodes (entre *peer* i *Tracker*) inclouen tota la informació corresponent, però aquesta s'ha triat que no sigui visible a través de l'aplicació ja que es va optar per usar *Wireshark* com a eina entremitja que capturés els missatges per visualitzar-los. Per tal de que *Wireshark* pugui entendre les dades destinades al protocol PPSP, degut a que aquest no és un estàndard oficial encara, s'ha desenvolupat un *protocol dissector* en Lua. De no haver programat aquest protocol dissector, *Wireshark* pot capturar les dades igualment, però no sabria reconèixer què és el que està capturant.

L'escenari pràctic que muntarem serà el que veurem a continuació. En aquest, el Peer *Android* estarà lligat als tres eixams existents, el *Peer 2* només al segon eixam i el *Peer 3* als eixams segon i tercer. D'aquesta forma podem veure diferents opcions i configuracions, i mostrar les captures corresponents.

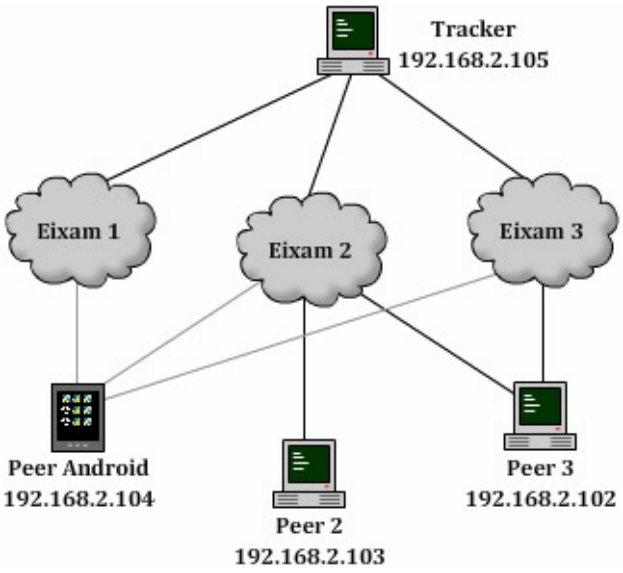

Fig. 7.26.- Esquema amb els diferents peers de cada eixam

Primer de tot, seguint una seqüència lògica de missatges, els tres *peers* han d'enviar els seus corresponents missatges de CONNECT cap al *Tracker*.

A les dues captures següents podem veure com és un missatge de tipus CONNECT enviat pel *Peer Android* amb el *Peer ID* amb valor *Peer Android*, i la resposta que li envia el *Tracker* en forma de missatge SUCCESFUL en aquest cas.

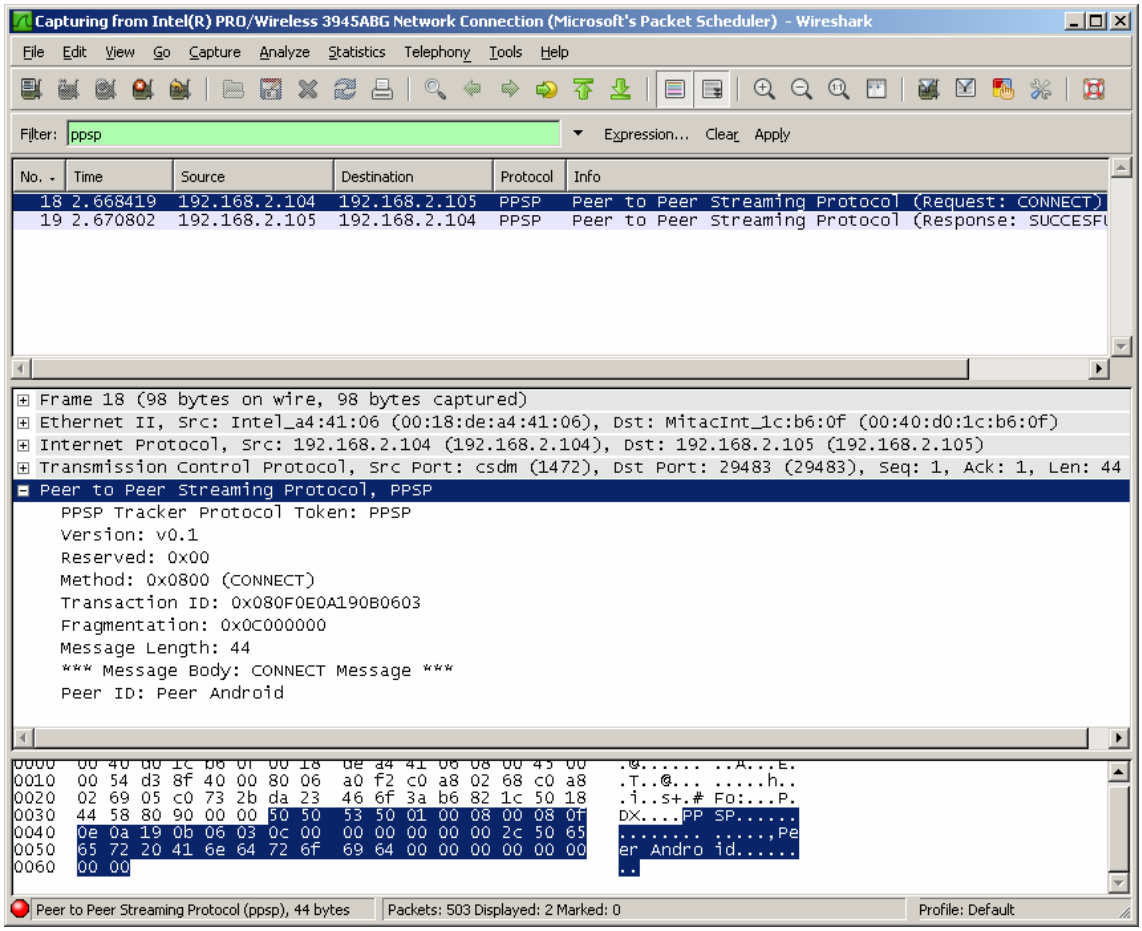

Fig. 7.27.- Captura de Wireshark d'una petició CONNECT

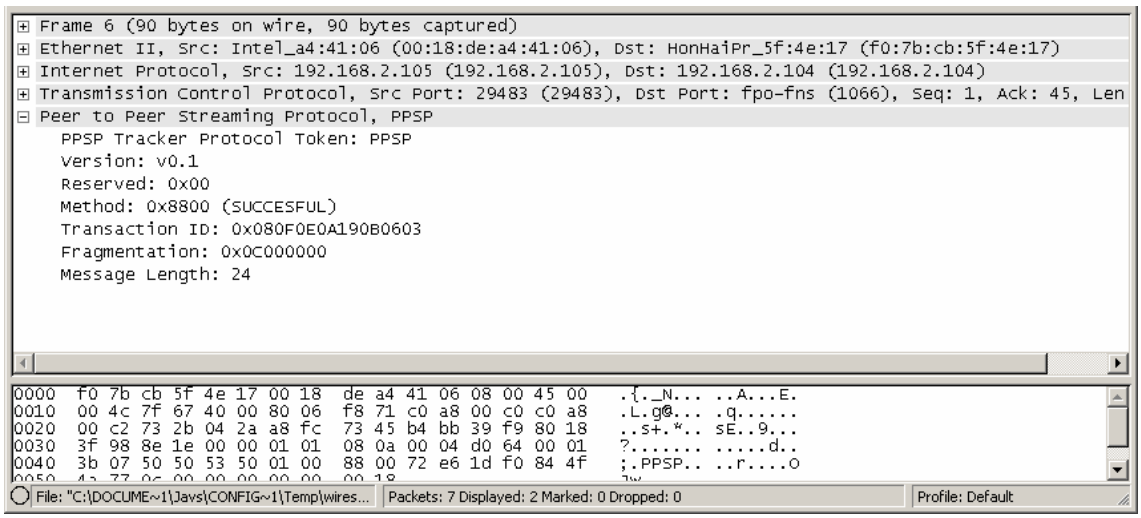

Fig. 7.28.- Captura de Wireshark d'una resposta SUCCESFUL

Tenint a punt l'eina per capturar els missatges, ben configurat el *protocol dissector* de PPSP a dins de la mateixa i tot l'escenari P2P de *peers* i *Tracker* amb les aplicacions iniciades i connectades entre elles, podem passar a veure com són les diferents captures dels missatges i quina informació es transmet.

Ja hem vist com és un missatge de tipus CONNECT i com són els missatges SUCCESFUL genèrics. De la resta de missatges que queden per veure, seguint una mica l'ordre de mètode que hi ha, començarem per muntar el sistema anterior, fent un CONNECT dels *peers* restants (*Peer 2* i *Peer 3*) i fent els JOIN (o JOIN_CHUNK) corresponents perquè cada peer entri als eixams desitjats. Per veure els dos tipus de missatge diferents, mostrarem un JOIN amb *Peer Android* i un JOIN_CHUNK amb *Peer 2*. La resta de missatges per unir-se als eixams seran iguals, amb el *Peer ID* corresponent.

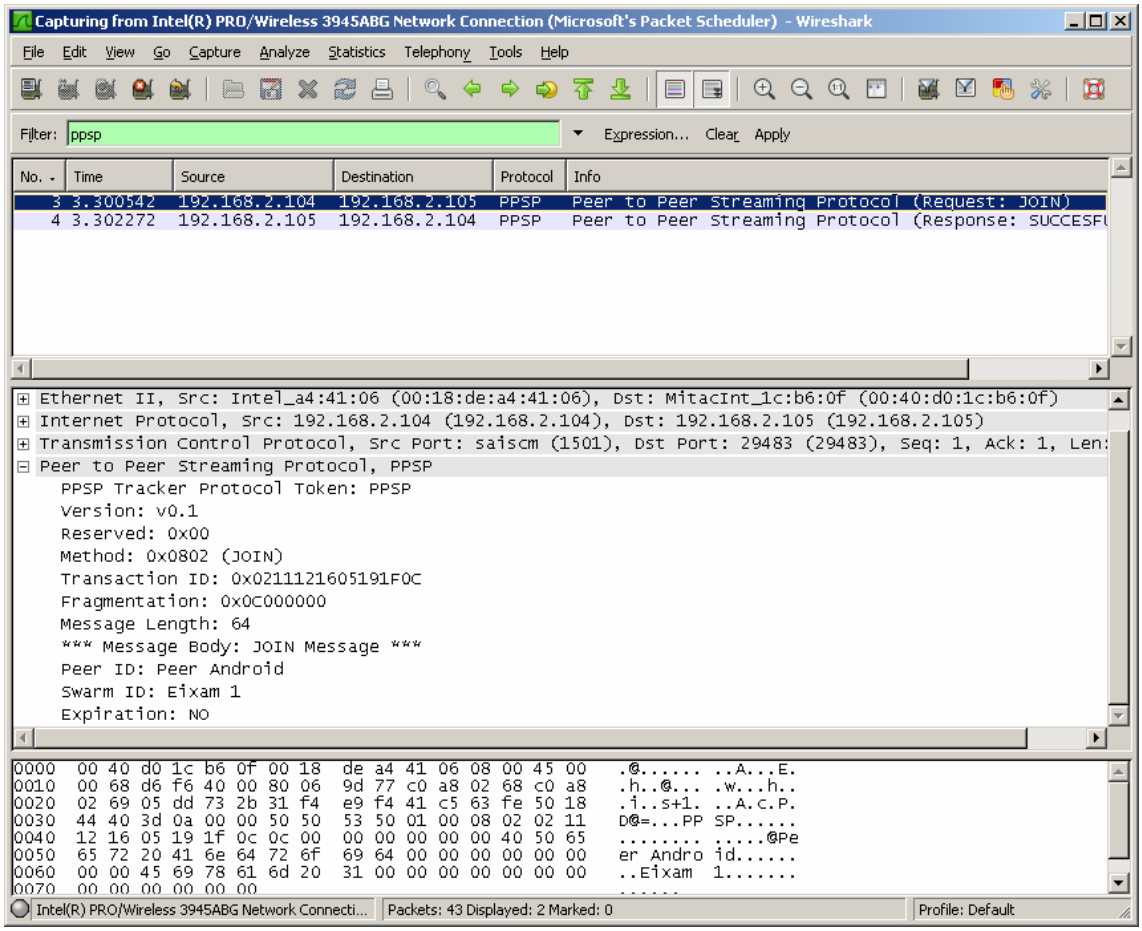

Fig. 7.29.- Captura de Wireshark del Peer Android fent una petició JOIN a l'Eixam 1

Es pot veure que l'identificador d'eixam al que vol pertànyer és *Eixam 1* (serà el mateix missatge per la resta) i que l'*Expiration Time* no hi és, és a dir, està posat a zero. Per poder fer missatges JOIN a la resta d'eixams, caldrà canviar l'identificador d'eixam als paràmetres de configuració.

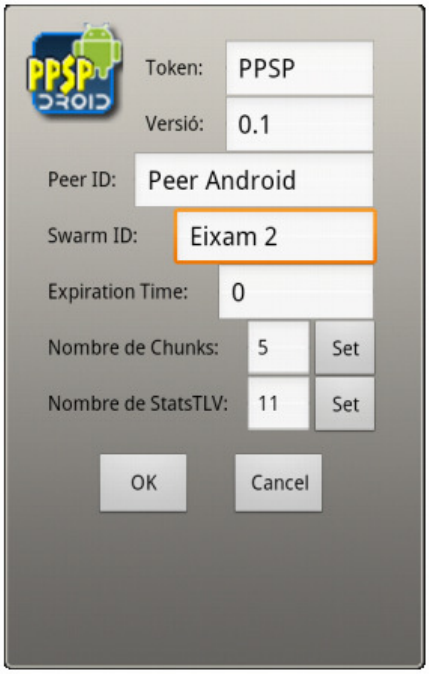

Fig. 7.30.- Selecció de l'eixam al que fer un JOIN o LEAVE al PPSP Droid Node

El JOIN_CHUNK el mostrem fet pel *Peer 2* i demanant formar part de l'*Eixam 2*.

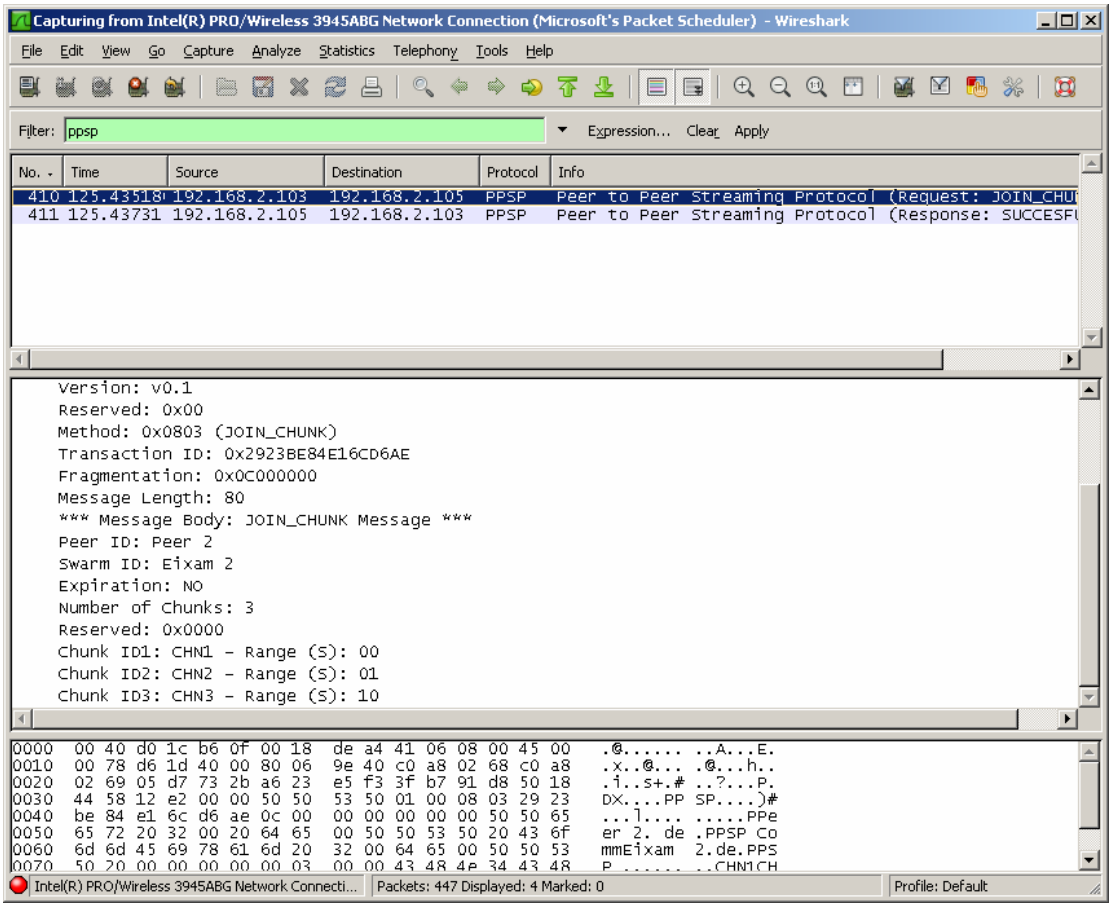

Fig. 7.31.- Captura de Wireshark del Peer 2 fent una petició JOIN_CHUNK a l'Eixam 2

Als missatges de JOIN_CHUNK, igual que passa amb els JOIN, s'ha d'indicar a quin eixam es vol unir el *peer*. A més, JOIN_CHUNK es diferència de JOIN per què al primer s'han d'indicar els *chunks* dels que disposa el *peer* per compartir a l'eixam.

A continuació, podríem mostrar captures de missatges tipus LEAVE i DISCONNECT que, tal i com hem vist a l'apartat del *PPSP Tracker Protocol*, són per abandonar un eixam o per fer una desconnexió del *Tracker*, respectivament. Tot i això, no és necessari mostrar-les ja que un missatge de tipus DISCONNECT és igual que un missatge CONNECT (inclou només el *Peer ID*), amb la diferència del ID de mètode. Pel cas de LEAVE és el mateix que JOIN, però per sortir d'un eixam en comptes d'entrar a formar-ne part. Per tant, havent vist les captures de *Wireshark* dels missatges CONNECT i JOIN, pels missatges DISCONNECT i LEAVE serien iguals. No serà necessari fer-les ja que no volem sortir dels eixams ni desconnectar. Arribats a aquest punt, ja tenim muntat l'esquema amb els tres *peers* connectats als diferents eixams del *Tracker*.

Ara, des del *Peer Android*, fem un missatge FIND per veure la resposta del *Tracker* i així comprovar quina llista de *peers* hi ha als diferents eixams. Fem un FIND al *Eixam 2*.

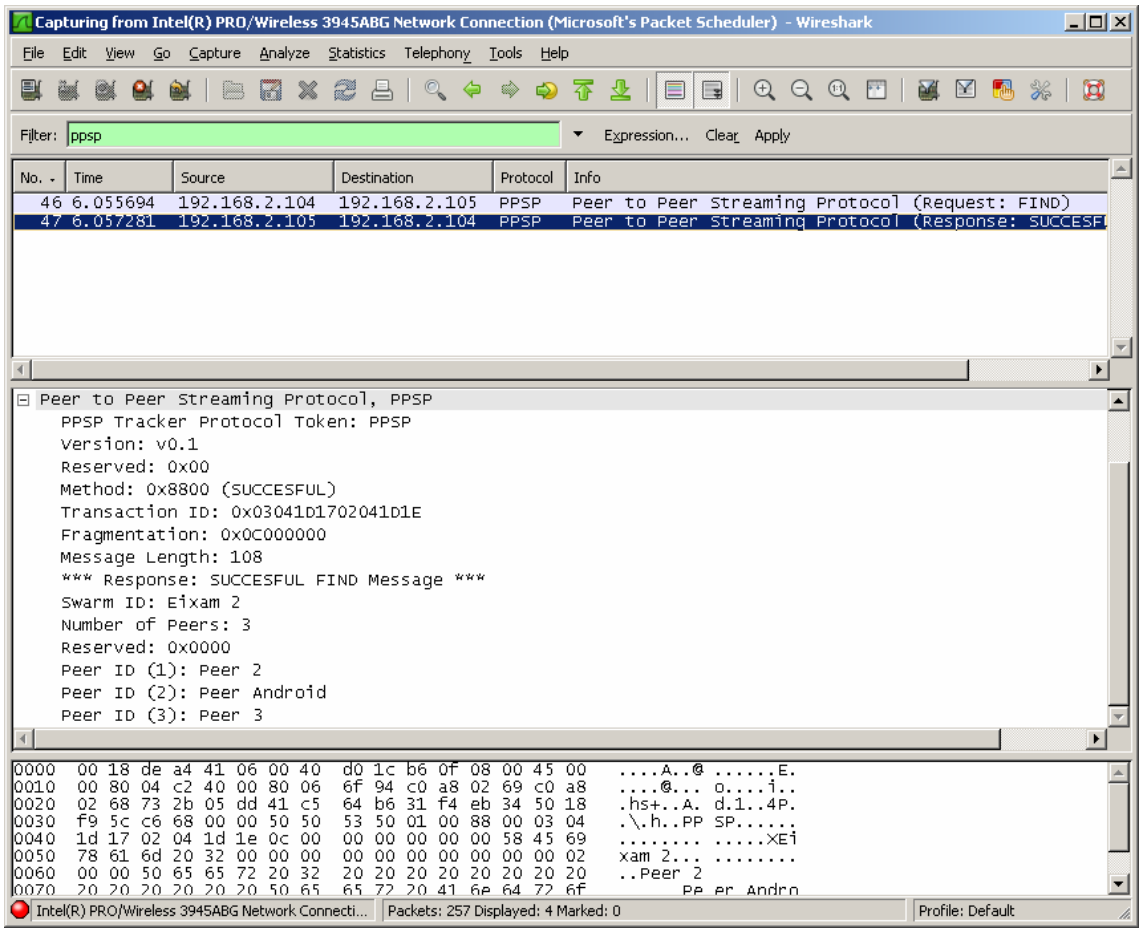

Fig. 7.32.- Captura de Wireshark d'una resposta FIND a l'Eixam 2

Es pot veure que a l'*Eixam 2* hi ha els tres *peers* coneguts llistats: *Peer Android*, *Peer 2* i *Peer 3*. En qualsevol cas, pels missatges FIND no val la pena mostrar les captures dels missatges de petició, ja que la informació important es troba a les respostes tal i com ja hem vist. Si féssim captures de les respostes obtingudes fent missatges FIND amb diferents identificadors d'eixam, veuríem que el *Tracker* ens retorna en tots els cassos les llistes de *peers* de cada eixam, igual que ha fet amb l'*Eixam 2*.

Aquestes captures les podem veure a continuació:

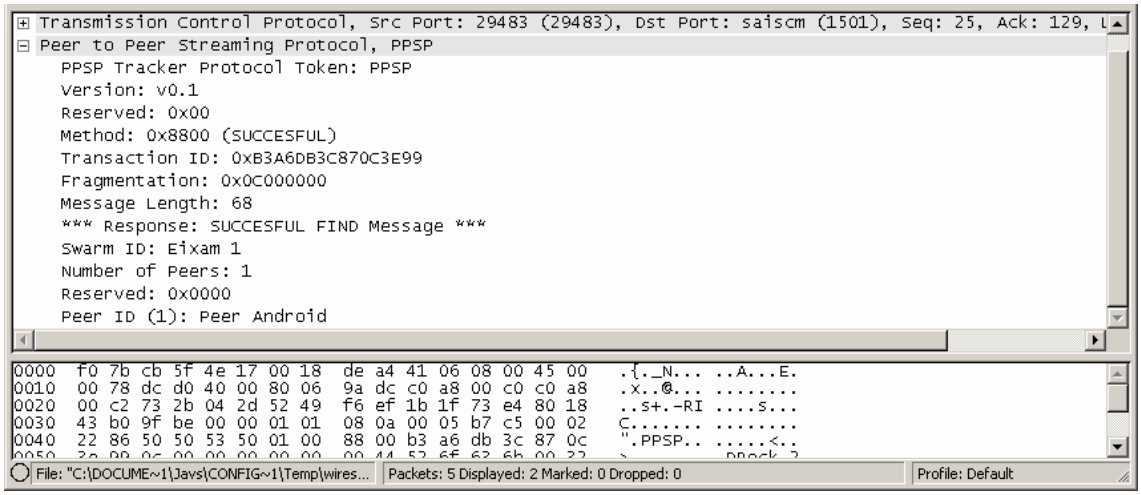

Fig. 7.33.- Captura de Wireshark d'una resposta FIND a l'Eixam 1

Es pot observar que tal i com està muntat l'escenari, a l'*Eixam 1* hi ha només el *Peer Android*. I, a la captura següent, veurem els peers que hi han llistats a l'*Eixam 3*, que seran *Peer Android* i *Peer 3*.

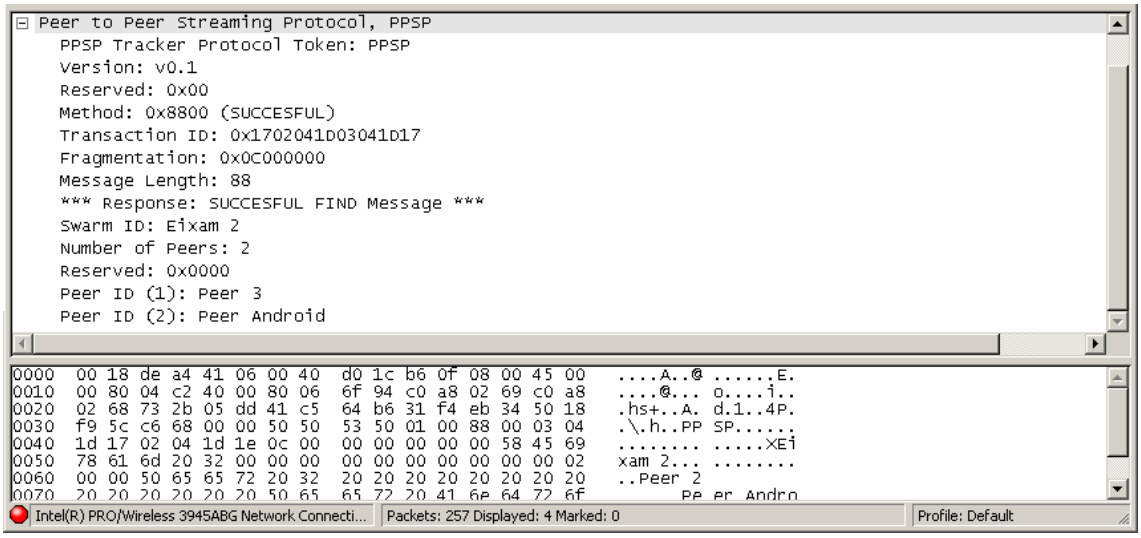

Fig. 7.34.- Captura de Wireshark d'una resposta FIND a l'Eixam 3

Per acabar les mostres dels resultats obtinguts a les captures, encara hem de veure la informació mostrada a un missatge de petició de tipus KEEPALIVE i els dos tipus de missatge estadístics. Un missatge de KEEPALIVE és útil en un escenari complet on les aplicacions tenen funcionalitat total, ja que serveix per anunciar periòdicament al *Tracker* que el *peer* que emet el missatge continua viu i, per tant, no l'ha d'esborrar de les llistes de *peers* corresponents a cada eixam on pertany el *peer*. Sobre el temps entre missatge i missatge d'aquest tipus, hem parlat a l'apartat *7.5. Estudi de la sobrecàrrega de la xarxa*.

En el nostre cas, les aplicacions, tot i complir amb el seu comés de les comandes, aquestes són independents entre elles i no hi ha control per part del *Tracker* sobre si un *peer* porta massa temps sense comunicar-se. Es podia haver optat per un *timer* que enviés aquesta comanda de forma periòdica en comptes de fer-ho manualment, que és com s'ha implementat, però no aporta cap millora al conjunt, degut a que és tot un entorn simulat.

Tot i això, la comanda és funcional i a la captura de la figura següent es poden veure les diferents dades que s'envien quan un *peer* (el *Peer Android* en aquest cas) fa una comanda de tipus KEEPALIVE cap al *Tracker*.

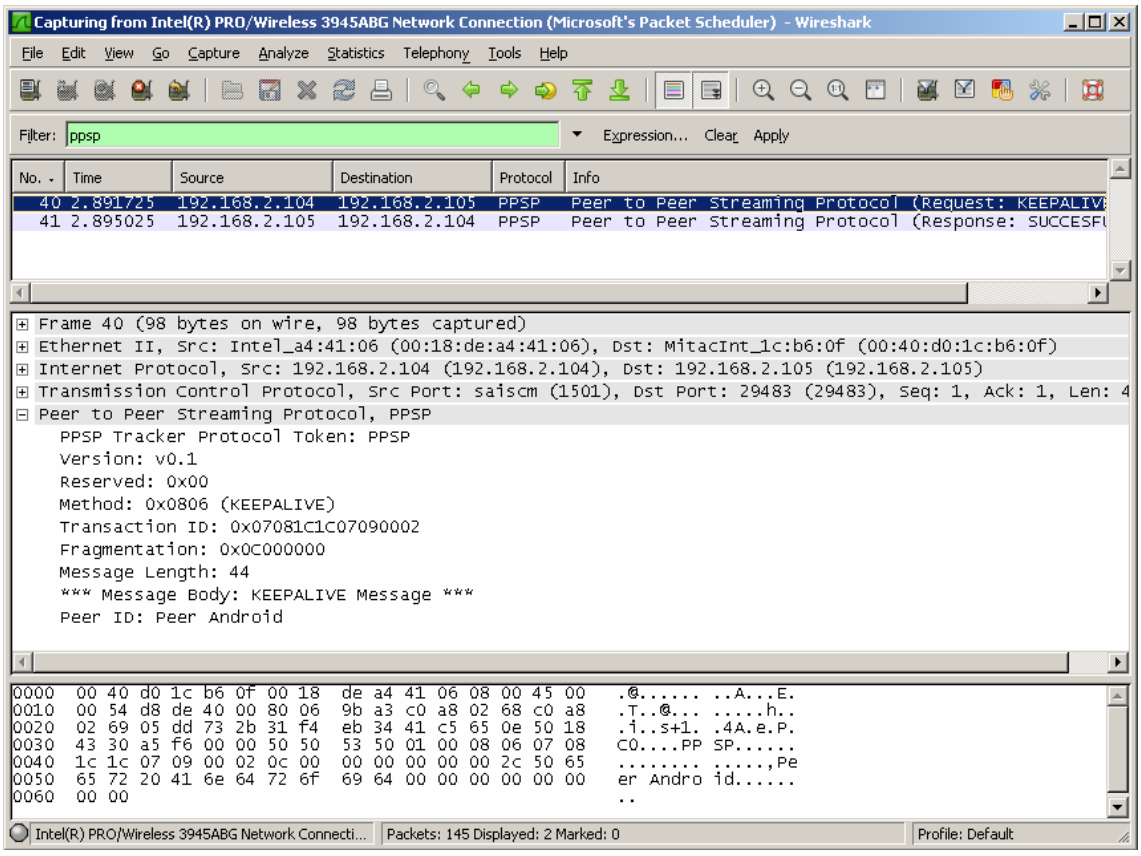

Fig. 7.35.- Captura a Wireshark d'una petició KEEPALIVE del Peer Android

D'altra banda, els missatges de peticions estadístiques són missatges que funcionen en ambdues direccions, és a dir, de *peer* cap a *Tracker* i de *Tracker* cap a *peer*. És per això que al *Tracker* es poden habilitar aquestes dues comandes (STAT_QUERY i STAT_REPORT) des del fitxer de configuració. En cas de no tenir-les habilitades, si un *peer* desitja demanar informació fent una petició d'aquest tipus, el *Tracker* respondrà amb un MESSAGE_NOT_SUPPORTED, indicant que no és un missatge que pugui processar. Al nostre cas, les funcions estadístiques estan habilitades al *Tracker*.

Abans de fer una petició estadística hem de tenir present quins són els diferents P-Types que volem enviar o dels que volem rebre informació quan demanem informació. Aquesta selecció la podrem canviar en tot moment i serveix tant per rebre informació com per enviar-la. En aquest cas, els marcarem tots.

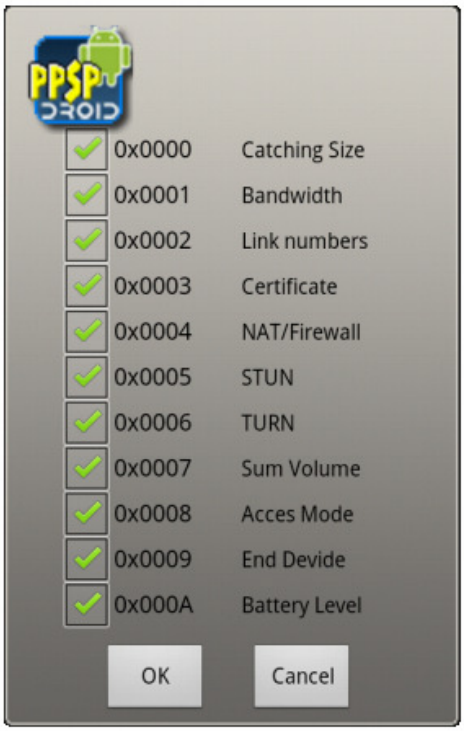

Fig. 7.36.- Configuració dels P-Types al PPSP Droid Node

Cal comentar també que, igual que succeeix amb els missatges KEEPALIVE, els missatges d'STAT_REPORT i STAT_QUERY, són missatges periòdics que s'envien cada cert temps per mantenir als *peers* i al *Tracker* informats contínuament sobre els canvis de la xarxa.

Un cop comentat això, podem veurem com és un missatge de petició d'STAT_QUERY enviat des del *Tracker* cap un *peer*. A la petició, el *Tracker* inclou al cos de missatge tot un seguit de *P-Types* (que hem indicat al fitxer de configuració inicial del mateix).

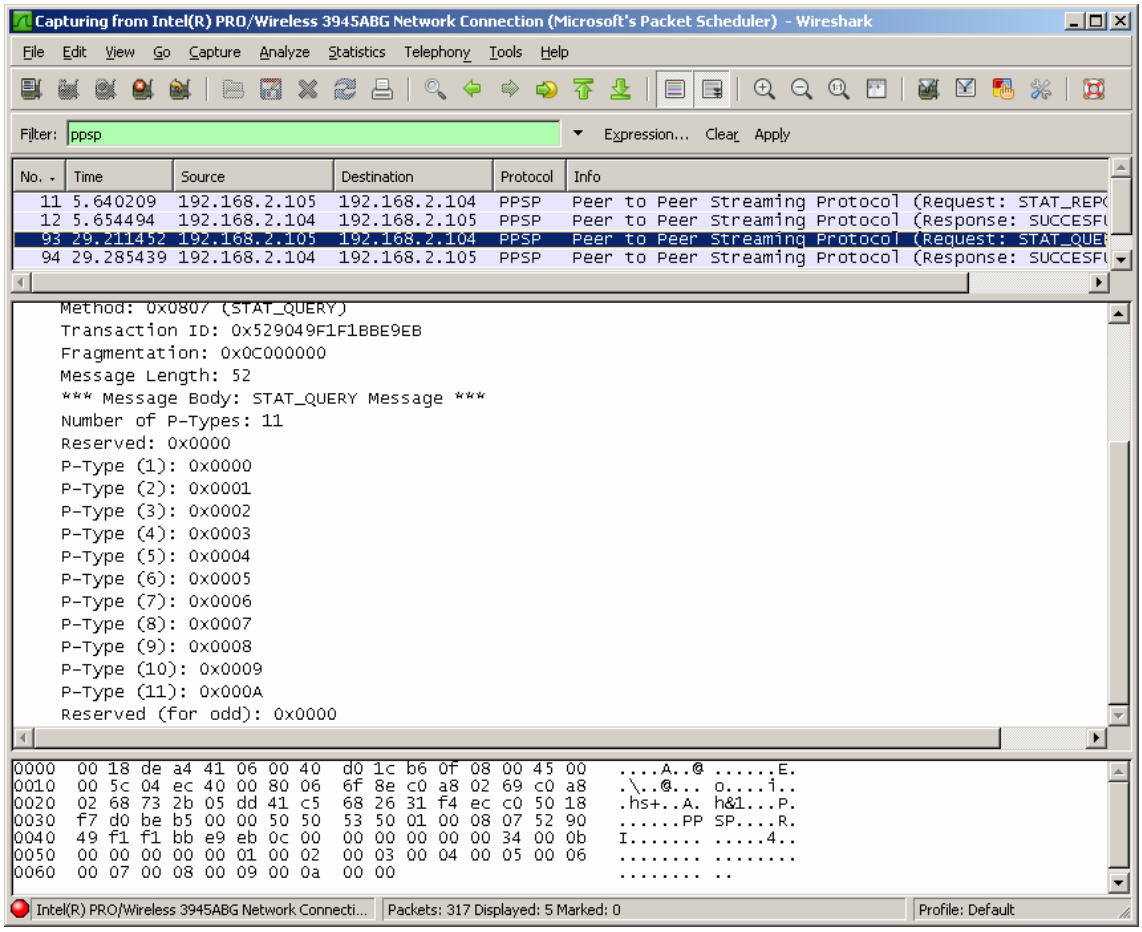

Fig. 7.37.- Captura de Wireshark d'una petició STAT_QUERY feta pel Tracker

Quan la petició la fa el *Tracker*, aquesta la reben tots els *peers*. Així el Tracker s'informa indistintament de tots els *peers* de la xarxa.

Els missatges de petició de tipus STAT_QUERY són, juntament amb les peticions de tipus FIND, els únics missatges que quan la resposta a la petició és SUCCESFUL el cos de missatge inclou dades. El cos de dades de la resposta de FIND ja hem vist que inclou la llista de *peers* completa de l'eixam pel que s'ha demanat.

En el cas de l'STAT_QUERY, la resposta inclourà un conjunt d'STAT TLVs, formats per diversos camps tal i com ja s'ha explicat a l'apartat *4. PPSP Tracker Protocol* d'aquesta mateixa memòria.

A la següent captura es pot veure amb detall com és aquesta resposta que dóna el *peer* a la petició feta pel *Tracker*. Els valors del camp *Property Value* dels STAT TLV són valors arbitraris posats per comprovar el funcionament dels missatges, els valors dels camps de *P-Type* els podem trobar a la seva taula corresponent i el camp *Length*, que indica la longitud del *Property Value* en bits, ha de ser múltiple de vuit (sinó internament es fa un *zero padding*).

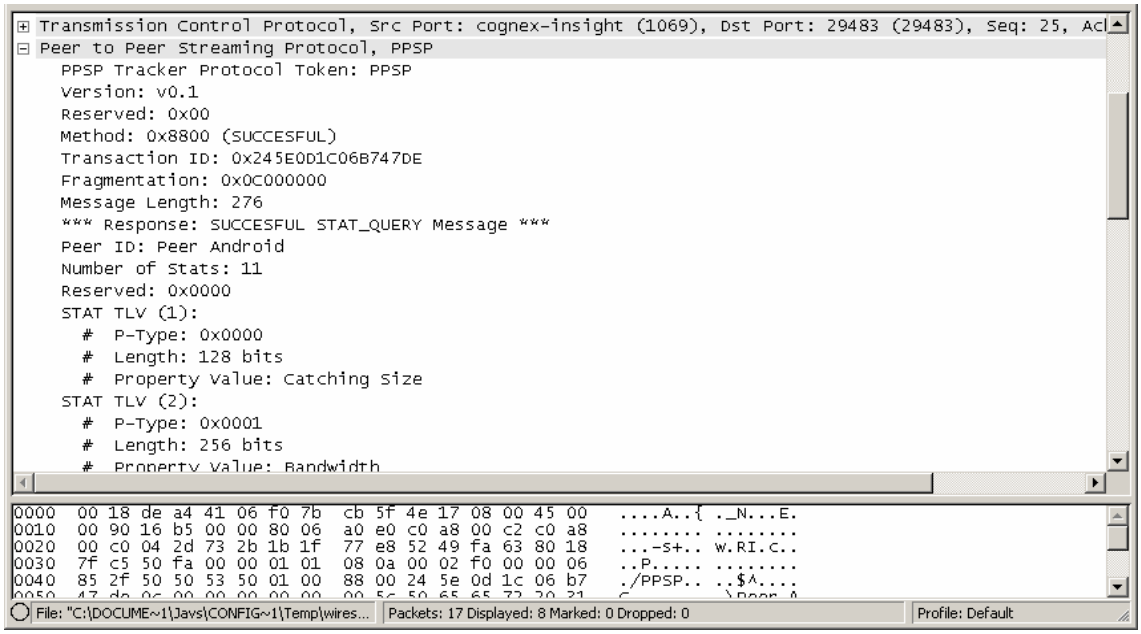

Fig. 7.38.- Captura de Wireshark d'una resposta STAT_QUERY feta cap al Tracker

En quant al missatge STAT_REPORT, el *Peer Android* fa una petició d'aquest tipus cap al *Tracker*. Igual que a la resposta enviada del *peer* al *Tracker* al missatge STAT_QUERY, a la petició STAT_REPORT el *peer* envia un seguit de STAT TLVs cap al *Tracker*.

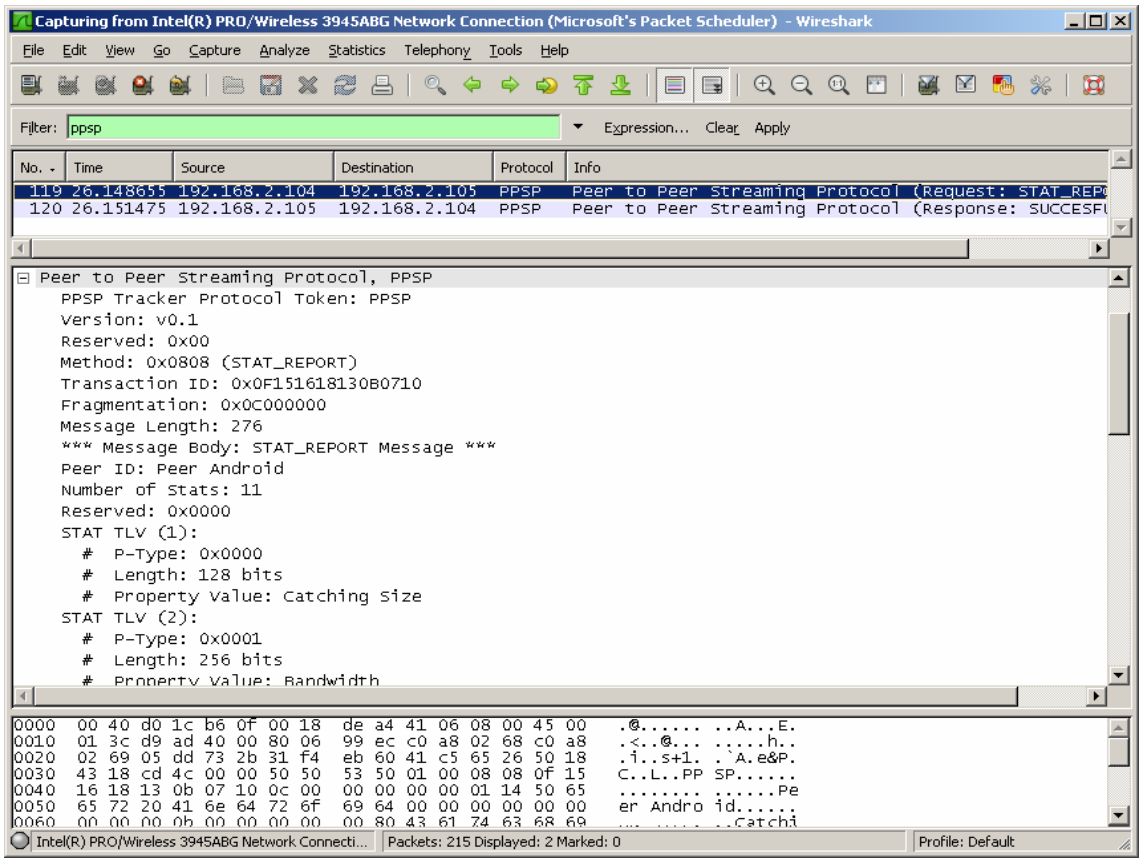

Fig. 7.39.- Captura de Wireshark d'una petició STAT_REPORT d'un peer

Per finalitzar la visualització de les captures dels diferents missatges, havent vist totes i cadascuna de les peticions i les diferents respostes SUCCESFUL que es poden rebre, quedaria per visualitzar les diferents respostes restants que indiquen algun problema.

Comencem per veure com seria una resposta a una petició amb sintaxi invàlida.

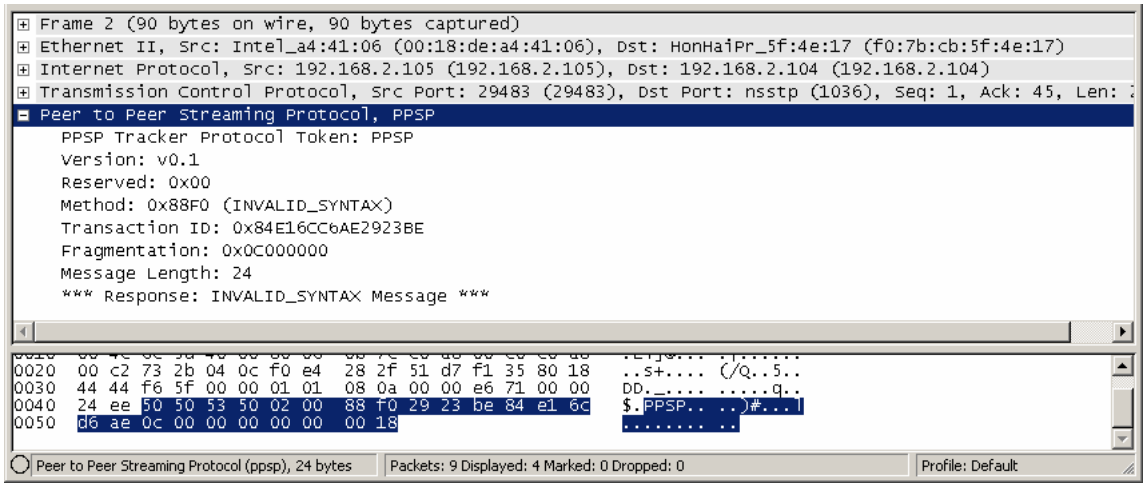

Fig. 7.40.- Captura de Wireshark d'una resposta INVALID_SYNTAX

Seguint l'ordre establert a la teoria, el següent missatge de resposta es tracta de VERSION_NOT_SUPPORTED, és a dir, una resposta a una petició amb versió incorrecta. L'única versió que suporta el protocol actualment és la 0.1. Qualsevol altra versió donarà com a resposta un missatge de versió no suportada.

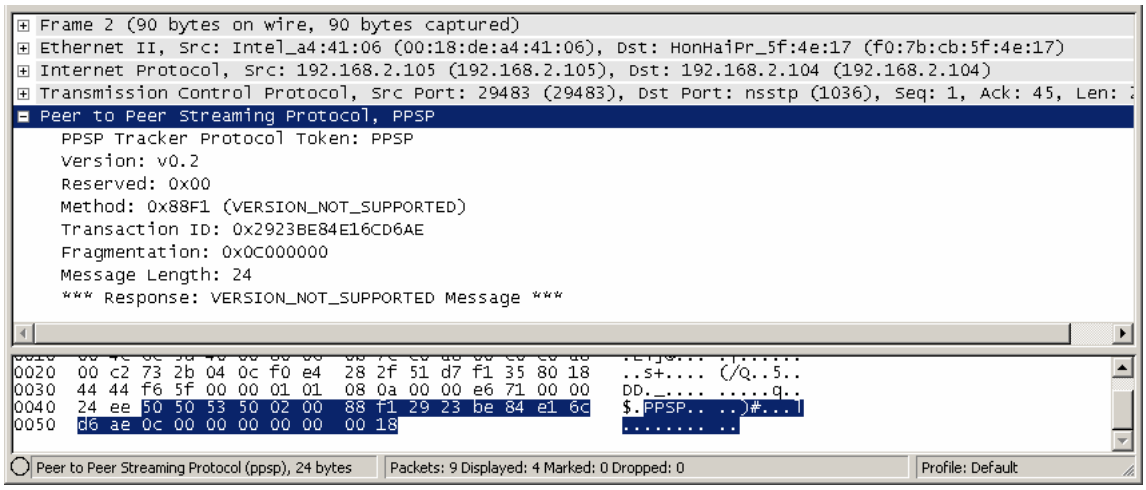

Fig. 7.41.- Captura de Wireshark d'una resposta VERSION_NOT_SUPPORTED

Tot seguit, forçarem un missatge de *Message Not Supported* com a resposta. Aquests missatges es donen quan una petició en particular no es suporta pel *Tracker*. A l'aplicació C++ hi ha dues formes d'obtenir aquest missatge com a resposta.

La primera d'aquestes és mitjançant la no habilitació de les funcions estadístiques per part del *Tracker*. La segona opció és a través de l'activació (posant a *true*) del booleà que es correspon amb aquest missatge des del fitxer de configuració inicial del *Tracker*.

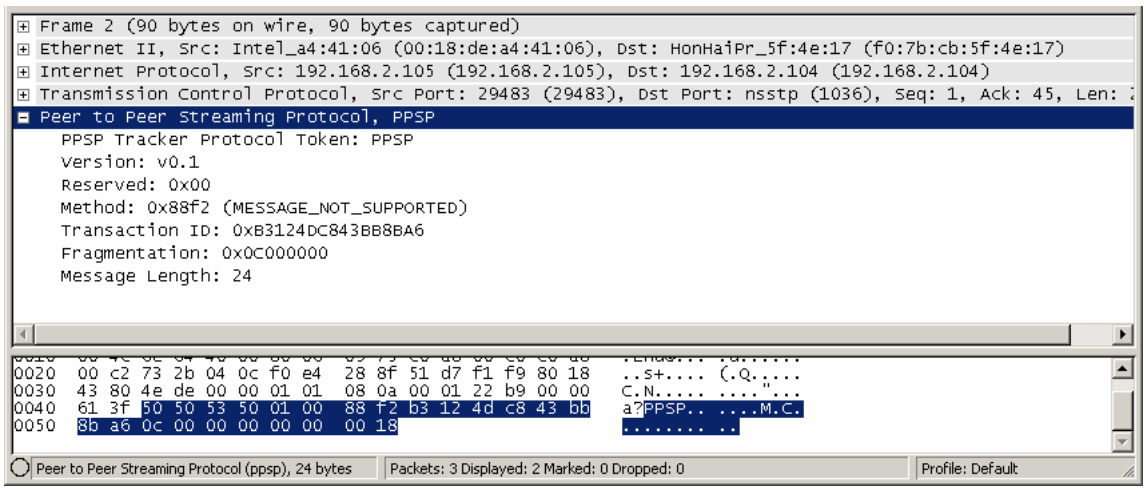

Fig. 7.42.- Captura de Wireshark d'una resposta MESSAGE_NOT_SUPPORTED

Un altre error que es pot forçar a la resposta que donarà el *Tracker* és el de *Temporarily Overloaded*. Aquest error indica que el *Tracker* està sobrecarregat i no pot processar la resposta. Degut a que el tràfic de dades de l'aplicació desenvolupada és molt petit, difícilment es podrà obtenir aquest missatge de forma natural, per això, igual que amb el missatge anterior, es pot forçar a que el *Tracker* doni aquest missatge si es desitja.

Habilitant el booleà corresponent al fitxer de configuració, es simularà la sobrecàrrega del *Tracker* de forma que respondrà en conseqüència.

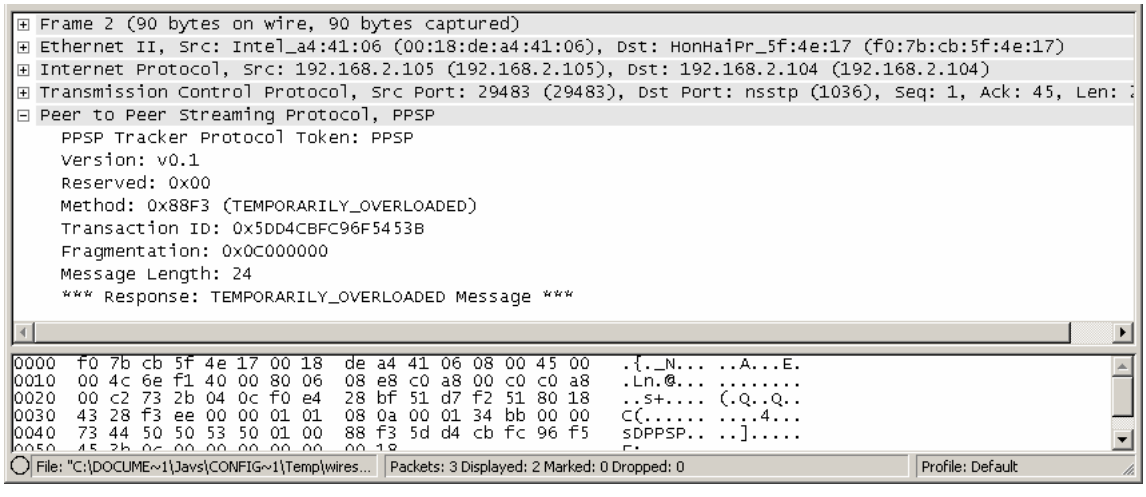

Fig. 7.43.- Captura de Wireshark d'una resposta TEMPORARILY_OVERLOADED

D'igual forma, si en comptes d'una sobrecàrrega del *Tracker* el que es vol simular és un error intern del mateix, per tal d'obtenir la resposta *Internal Error*, es pot fer de la mateixa forma amb el booleà corresponent.

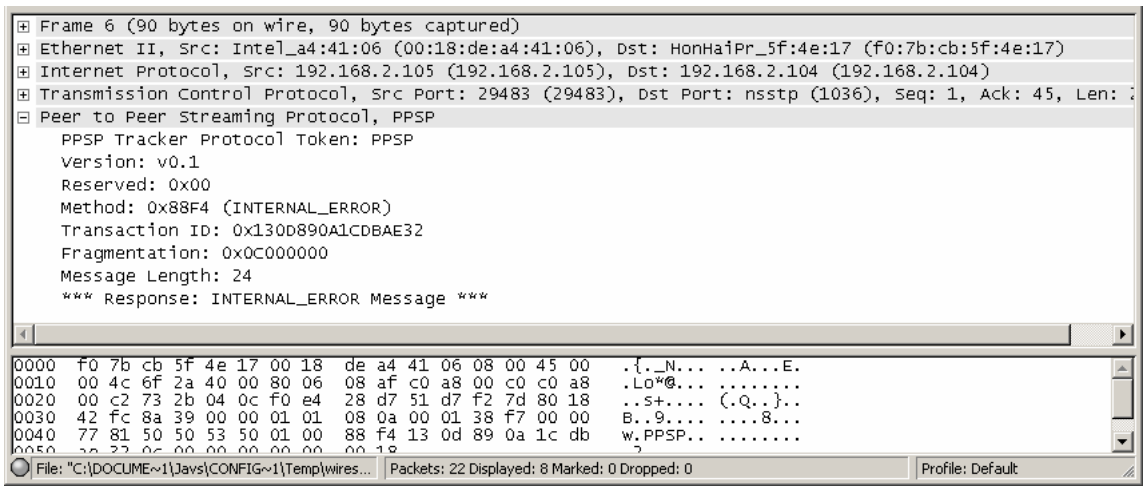

Fig. 7.44.- Captura de Wireshark d'una resposta INTERNAL_ERROR

Es pot donar el cas de que algunes peticions no estiguin permeses pel *Tracker*. El missatge de resposta que s'emet és un amb mètode MESSAGE_FORBIDDEN. Com en casos anteriors, la forma de visualitzar aquesta variant és de forma manual amb el booleà corresponent. Forçant aquest a *true*, podem comprovar com el *Tracker* ens respon a les peticions indicant que el missatge està prohibit.

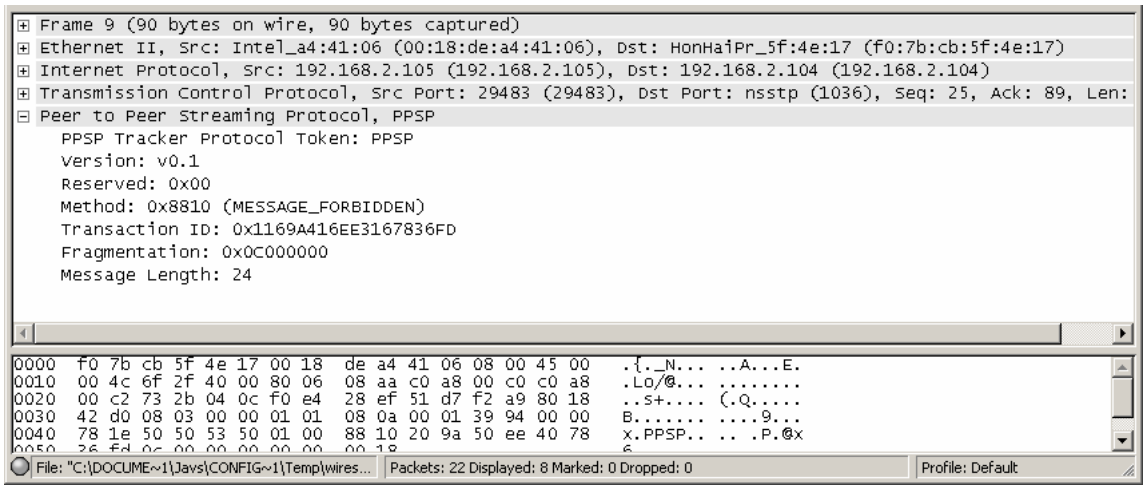

Fig. 7.45.- Captura de Wireshark d'una resposta MESSAGE_FORBIDDEN

Per poder observar un missatge de OBJECT_NOT_FOUND, cal fer una petició FIND demanant per algun eixam no existent. Aquesta resposta, donada als missatges de petició de tipus FIND, ens indica que l'objecte pel que s'ha demanat no existeix a la base de dades del *Tracker*.

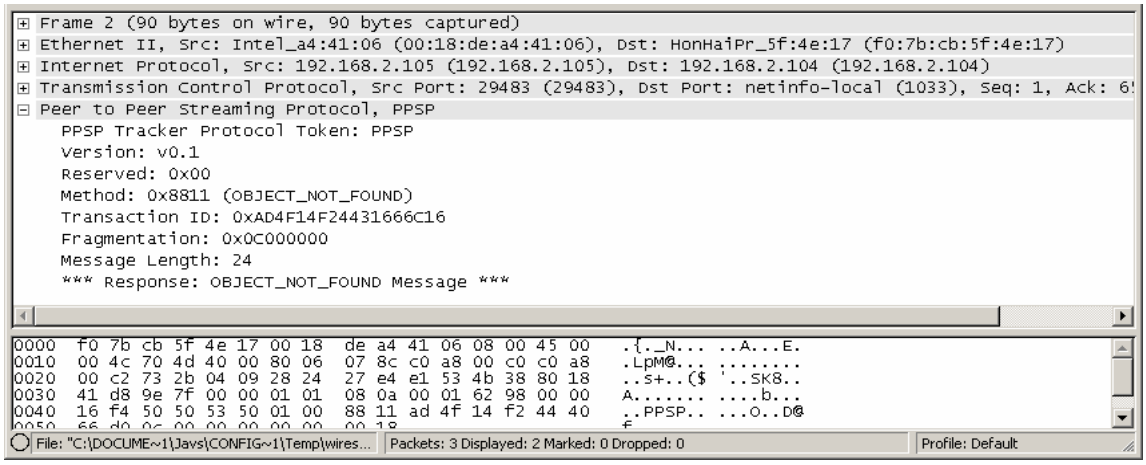

Fig. 7.46.- Captura de Wireshark d'una resposta OBJECT_NOT_FOUND

Finalment, en un entorn real, es pot donar el cas de que el *Tracker* sigui privat i requereixi autentificació i/o pagament per usar els seus serveis. Es poden representar amb uns booleans configurables AUTHENTITACION_REQUIRED i PAYMENT_REQUIRED.

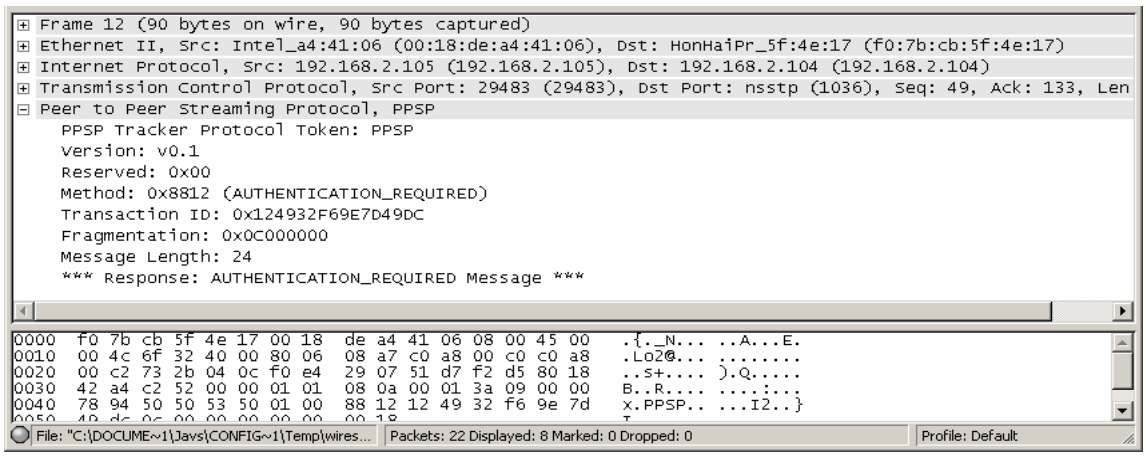

Fig. 7.47.- Captura de Wireshark d'una resposta AUTHENTICATION_REQUIRED

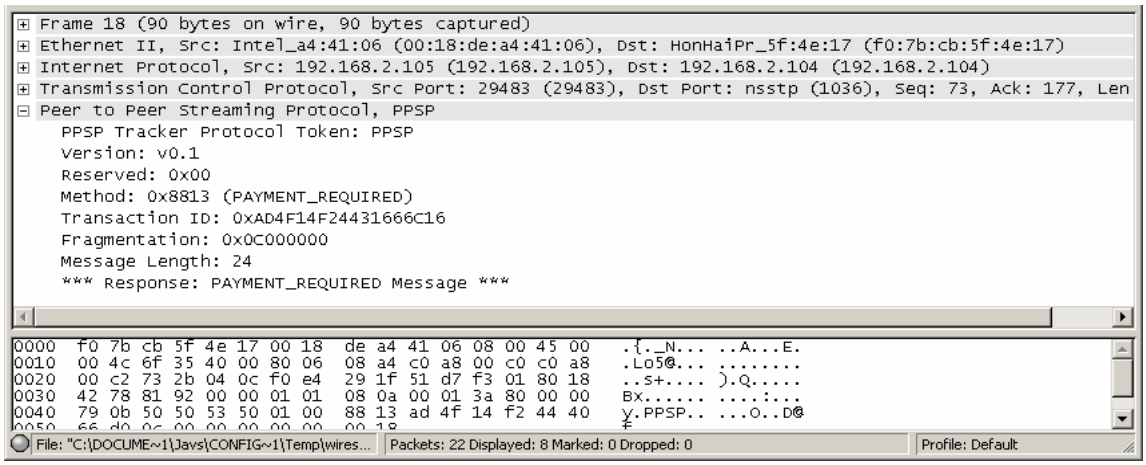

Fig. 7.48.- Captura de Wireshark d'una resposta PAYMENT_REQUIRED

8. Conclusions i línies de futur

PPSP és un protocol que encara està en desenvolupament i, si bé és cert que ja s'ha avançat bastant, encara queda molt de camí que recórrer. De fet, no serà fins l'agost de 2011 quan està proposat des del propi *working group* la presentació del *PPSP Tracker Protocol* i el *PPSP Peer Protocol* com a proposta d'estàndard al IESG[45], i no serà fins decembre d'aquest mateix any que està previst presentar la guia d'us al IESG. Tot i que encara queda molt per davant en quant a l'estandardització del protocol, els estàndards definits fins ara van prenent forma i es pot començar a treballar amb ells.

En aquest projecte s'ha treballat amb aquesta idea, partint de l'estudi general del protocol (així com dels protocols que inclou) fet al projecte anterior, ampliant aquesta base amb les actualitzacions i millores actuals, amb una introducció amplia sobre el *Peer to Peer Streaming* de forma general. Així, cal fer una diferenciació entre aquest treball i l'anterior projecte, ja que tot i que es tracta el mateix tema, al projecte anterior es feia una introducció i estudi del protocol que s'ha ampliat amb noves característiques en aquest treball.

A més d'incidir més en el que és el PPSP, deixant de banda les explicacions més generals sobre P2P i *streaming* P2P, aquest treball parla d'*streaming* P2P per nodes mòbils i inclou el desenvolupament d'una eina programada amb *Android*. Així doncs, a més d'aquesta ampliació de l'estudi de la que es parla, el projecte consta de la implementació (a una eina funcional programada en *Android*) del *PPSP Droid Node*, complementari al *PPSP Node Communication* (programat en C++), que s'encarrega de la comunicació de senyalització entre *Tracker* i *peers* a un sistema *streaming* P2P funcionant sota PPSP; complementada amb el l'ús del *protocol dissector* de PPSP per *Wireshark* prèviament desenvolupat.

La part més important d'aquesta aplicació desenvolupada per *Android*, igual que ho va ser a l'eina programada en C++, ha sigut el desenvolupament de la classe que s'encarrega de generar els diferents missatges del *PPSP Tracker Protocol*, adaptada a *Java* de la classe en C++, així com tot l'estudi i el desenvolupament per *Android* que és una tecnologia molt nova, ja que s'ha conformat com un mòdul individual que es podria implementar a dins de qualsevol altra aplicació que vulgui fer servir aquests missatges usant el protocol.

A més, conjuntament amb aquesta eina que fa les connexions entre un *peer* i el *Tracker*, i l'enviament dels missatges del protocol, es fa servir el *protocol dissector* de PPSP per *Wireshark* programat en llenguatge Lua desenvolupat prèviament a l'anterior

projecte, permetent així a l'analitzador de xarxes veure correctament els paquets referents a PPSP al capturar-los. Això és necessari ja que d'entrada, *Wireshark* no pot reconèixer aquests paquets com a tals, ja que el protocol encara no s'implementa enlloc de forma oficial.

Els diferents components mencionats, tant la classe per implementar els missatges de PPSP (per C++ i *Android*) i el *protocol dissector* de PPSP per *Wireshark*, juntament amb tot l'estudi teòric i matemàtic realitzat, són els valors afegits del projecte i el treball, podent-se fer servir a futures ampliacions del mateix o a altres projectes relacionats amb el protocol, ja que els dos funcionen com a mòduls individuals independents. S'ha de tenir present, però, que ambdós components estan programats sense tenir en compte el *PPSP Peer Protocol*, ja que encara s'està treballant des del *working group* a la seva definició.

Per tant, una primera revisió del projecte consistiria en afegir els missatges d'aquest protocol un cop definits i, de pas, ampliar l'aplicació per què hi hagués comunicació entre *peers* i no només entre *peers* i *Tracker*, formant així un sistema complet de senyalització *peer to peer* en *streaming*.

Anant un pas més enllà, un cop es tinguessin implementats el *PPSP Tracker Protocol* i el *PPSP Peer Protocol*, es podria aprofundir més i complementar tot l'escenari amb un petit sistema d'intercanvi d'arxius, que funcionaria sota algun protocol de transport i, d'aquesta forma, es tindria un escenari complet, amb tota la part de senyalització i de transport implementades.

A més, es podrien fer modificacions per tal de que els dispositius mòbils puguin funcionar fent servir les connexions 3G o 4G per connectar-se a la xarxa, ja que ara l'aplicació no comtempla connexions entre xarxes diferents i, per tant, només es pot fer servir el WiFi del dispositiu per connectar-lo a la mateixa xarxa. En aquest mateix punt, es pot parlar sobre afegir NAT o fins i tot un sistema de VPN per l'aplicació pels nodes fixes, per poder funcionar entre diferents xarxes i així també es podrien fer servir les connexions d'Internet mòbil dels terminals *Android*.

D'altra banda, centrant-nos més en el protocol en sí, PPSP suposa un pas endavant en quant a l'*streaming peer to peer*. L'elaboració d'un estàndard no propietari dins d'aquest camp, pot suposar un avanç ja que qualsevol aplicació tindrà possibilitat d'oferir serveis d'aquest tipus sense necessitat de desenvolupar un protocol propi, partint de l'estàndard. Fins ara, l'únic que hi havia eren protocols propietaris desenvolupats per aplicacions concretes, no compatibles entre sí.

En definitiva, un estàndard fa que la compatibilitat entre aplicacions sigui màxima, podent integrar millor funcions en comú i així poder oblidar-se d'incompatibilitats per poder centrar-se en altres temes.

Si bé és cert que encara no està aprovat oficialment el protocol, el camí que porta, gràcies al gran treball que es fa des del *working group* del IETF format per tal efecte, farà que el protocol PPSP sigui segurament el futur per l'*streaming peer to peer*, conformant un estàndard sòlid i complet. Tot i això, encara queden molts temes oberts que s'han de tractar, com la seguretat i altres consideracions del protocol, i serà durant els propers mesos quan s'aniran perfilant aquest i les seves característiques i funcionalitats.

9. Bibliografia

[1] "Streaming Media", <http://en.wikipedia.org/wiki/Streaming_media>, [Consultada a Febrer 2011]

[2] Schollmeier, R., "A definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications", Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE (2002)

[3] Zhang, Y., Jennings, C., and Eggert, Lars, "Peer to Peer Streaming Protocol (PPSP)", <https://dataTracker.ietf.org/wg/ppsp/charter/>, [Consultada a Octubre 2009 - Setembre 2010]

[4] Internet Engineering Task Force (IETF), a: <http://www.ietf.org>, [Consultada a Octubre 2009 - Setembre 2010]

[5] Request for Comments (RFC), a: <http://www.ietf.org/rfc.html> [Consultada a Agost-Setembre 2010]

[6] Zhang, Y., Zong, N., Camarillo, G., Seng, J., Yang, R., "Problem Statement of PPSP", Gener 2011, <draft-ietf-ppsp-problem-statement>

[7] CCTV, <http://en.wikipedia.org/wiki/Closed-circuit_television>, [Consultada a Abril 2011]

[8] PPLive, <http://en.wikipedia.org/wiki/PPLive>, [Consultada a Abril 2011]

[9] PPStream, <http://en.wikipedia.org/wiki/PPStream>, [Consultada a Maig 2011]

[10] Seedorf, J., and Burger, E., "Application-Layer Traffic Optimization (ALTO) Problem Statement", <draft-marocco-alto-problem-statement>, Març 2009

[11] Schulzinne, H., Rao, A., and Lanphier, R., "Real Time Streaming Protocol (RTSP)", RFC 2326, Abril 1998

[12] Gu, Y., Bryan, D., Zhang, Y., and H. Liao, "PPSP Tracker Protocol", Març 2010, <draft-gu-ppsp-Tracker-protocol>

[13] Gu, Y., and Bryan, D., "PPSP Peer Protocol", Juliol 2010, <draft-gu-ppsp-peer-protocol>

[14] Mobile IP Protocol (MIP), <http://www.tcpipguide.com/free/t_InternetProtocolMobilitySupportMobileIP.htm>, [Consultada a Febrer 2011]

[15] Johnson, D., Perkins, C. and Arkko. J,. "Mobility Support in IPv6", <RFC 3775>, Juliol 2004

[16] Gundavelli, S., Devarapalli, V., Chowdhury, K., and Patil, B., "Proxy Mobile IPv6", <RFC 5213>, Agost 2008

[17] Moskowitz, R., Nikander, P., Jokela, P., and Henderson, T., "Host Identity Protocol", <RFC 5021>, Abril 2008

[18] "Internet Research Task Force (IRTF)", <http://www.irtf.org/>, [Consultada a Març 2011]

[19] Camarillo, G., Nikander, P., Hautakorpi, J., Keranen, A., and Johnston, A., "Host Identity Protocol (HIP) Based Overlay Networking Environment (BONE)", RFC 6079, Gener 2011

[20] Wireshark, <http://www.wireshark.org>, [Consultada a Juny-Setembre 2010]

[21] "Lua: The programming language", <http://www.lua.org/>, [Consultada a Juny-Setembre 2010]

[22] "Android", <http://www.android.org/>, [Consultada a Febrer 2011]

[23] "Linux", <http://www.linux.com>, [Consultada a Febrer 2011]

[24] "Google", <http://www.google.com/>

[25] "Java", <http://www.java.com>, [Consultada a Març 2011]

[26] "Apache", <http://www.apache.org>, [Consultada a Abril 2011]

[27] "Symbian Foundation", <http://www.symbian.org>, [Consultada a Abril 2011]

[28] "Apple", <http://www.apple.com>, [Consultada a Març 2011]

[29] "Dalvik Virtual Machine", <http://www.dalvikvm.com>, [Consultada a Març 2011]

[30]" Scalable Graphics Library (SGL)", <http:// en.wikipedia.org/wiki/SGL >, [Consultada a Març 2011]

[31]" Secure Sockets Layer (SSL)", <http://www.iec.csic.es/criptonomicon/ssl.html>, [Consultada a Abril 2011]

[32] "Open Handset Alliance (OHA)", <http://www.openhandsetalliance.com/>, [Consultada a Febrer 2011]

[33] "Federal Communications Commission (FCC)", <http://www.fcc.gov/>, [Consultada a Abril 2011]

[34] "Near Field Communication (NFC)", <http://www.nfc-forum.org/>, [Consultada a Febrer 2011]

[35] iOS, <http://es.wikipedia.org/wiki/IOS_%28sistema_operativo%29>, [Consultada a Març 2011]

[36] Windows Phone, <http://es.wikipedia.org/wiki/Windows_Phone>, [Consultada a Març 2011]

[37] "Sun microsystems", <http://www.sun.com>, [Consultada a Març 2011]

[38] Free Software Foundation, "GNU General Public License (GPL)", <http://www.gnu.org/copyleft/gpl.html>, Juny 2007

[39] "Endianness, Little endian", "http://en.wikipedia.org/wiki/Endianness", [Consultada a Març 2011]

[40] "Endianness, Big endian", "http://en.wikipedia.org/wiki/Endianness", [Consultada a Març 2011]

[41] Strauss, F., Ransom, S., Lahde, S. and Wellnitz, O., "P2P Chat Protocol", <draft-strauss-p2p-chat>, Abril 2007

[42] Xiao, L., Bryan, D.A., Gu, Y. and Tai, X., "A PPSP Tracker Usage for Reload", <draft-xiao-ppsp-reload-distributed-tracker>, Febrer 2011

[43] Big O notation, a: <http://en.wikipedia.org/wiki/Big_O_notation> [Consultada a Maig 2011]

[44] Handley,M. and Perkins, C., <SAP: Session Announcement Protocol>, <draft-ietf-mmusic-sap-v2>, Octubre 2000

[45] "Internet Engineering Steering Group (*IESG*)*", <http://www.ietf.org/iesg>, [Consultada a Maig 2011]*

10. Annexos

10.1. Classe per generar missatges PPSP Tracker Protocol

```
package ppsp.droid;

import android.util.Log;

@SuppressWarnings("unused")
public class PPSPMessages {

	/*Variables*/

	public PPSPMessages() {}

	public byte[] HeaderMessage(String strToken,char cVersion,char cReserved,String strMethod,byte[]
byTransaction,byte[] byFragmentation,int nMessageLength)
	{
	byte[] byAux;
	byte[] byHeader;
	char c;
	int i;

	byHeader = new byte[24];

	//PPSP Token
	byAux = new byte[4];
	byAux = strToken.getBytes();		//0-3
	for (i=0;i<4;i++)
		byHeader[i]=byAux[i];

	//Versio
	byHeader[4] = (byte) cVersion;

	//Reserved
	byHeader[5] = (byte) cReserved;

	//Method
	byAux = null;
	byAux = new byte[2];
	byAux = GetMethod(strMethod);

	byHeader[6] = byAux[0];
	byHeader[7] = byAux[1];

	//Transaction ID
	for (i=0;i<8;i++)
		byHeader[i+8] = byTransaction[i];

	//Fragmentation
	for (i=0;i<4;i++)
		byHeader[i+16] = byFragmentation[i];

	//Length
	byHeader[20] = (byte) (0xff & (nMessageLength>>24));
	byHeader[21] = (byte) (0xff & (nMessageLength>>16));
	byHeader[22] = (byte) (0xff & (nMessageLength>>8));
	byHeader[23] = (byte) (0xff & (nMessageLength));

	return byHeader;
	}

	/*******************************/
	//Request Messages
```

```
	public byte[] ConnectMessage (String strPeerID)
	{
		byte[] byBody;
		String strBody,aux;
		char c = 0x00;

		byBody = new byte[20];
		strBody = strPeerID;
		aux = String.valueOf(c);

		while (strBody.length()<20)
			strBody = strBody.concat(aux);

		byBody = strBody.getBytes();

		return byBody;
	}

	public byte[] DisconnectMessage (String strPeerID)
	{
		byte[] byBody;
		String strBody,aux;
		char c = 0x00;

		byBody = new byte[20];
		strBody = strPeerID;
		aux = String.valueOf(c);

		while (strBody.length()<20)
			strBody = strBody.concat(aux);

		byBody = strBody.getBytes();

		return byBody;
	}

	public byte[] JoinMessage(String strPeerID,String strSwarmID,int nExpirationTime)
	{
		byte[] byBody;
		String strBody,aux;
		char c = 0x00;

		byBody = new byte[40];
		strBody = strPeerID;
		aux = String.valueOf(c);

		while (strBody.length()<20)
			strBody = strBody.concat(aux);

		strBody = strBody + strSwarmID;

		while (strBody.length()<20+16)
			strBody = strBody.concat(aux);

		strBody = strBody.concat(String.valueOf((char)(0xff & (nExpirationTime>>24))));
		strBody = strBody.concat(String.valueOf((char)(0xff & (nExpirationTime>>16))));
		strBody = strBody.concat(String.valueOf((char)(0xff & (nExpirationTime>>8))));
		strBody = strBody.concat(String.valueOf((char)(0xff & (nExpirationTime))));

		byBody = strBody.getBytes();

		return byBody;
	}

	public byte[] JoinChunkMessage(String strPeerID,String strSwarmID,int nExpirationTime,int
nNumChunks,String strChunks)
```

```
{
        byte[] byBody;
        String strBody,aux;
        char c = 0x00;

        byBody = new byte[40];
        strBody = strPeerID;
        aux = String.valueOf(c);

        while (strBody.length()<20)
                strBody = strBody.concat(aux);

        strBody = strBody + strSwarmID;

        while (strBody.length()<20+16)
                strBody = strBody.concat(aux);

        strBody = strBody.concat(String.valueOf((char)(0xff & (nExpirationTime>>24))));
        strBody = strBody.concat(String.valueOf((char)(0xff & (nExpirationTime>>16))));
        strBody = strBody.concat(String.valueOf((char)(0xff & (nExpirationTime>>8))));
        strBody = strBody.concat(String.valueOf((char)(0xff & (nExpirationTime))));

        //Number of Chunks
        strBody = strBody.concat(String.valueOf((char)(0xff & (nNumChunks>>8))));
        strBody = strBody.concat(String.valueOf((char)(0xff & (nNumChunks))));

        //Reserved
        strBody = strBody.concat(aux);
        strBody = strBody.concat(aux);

        //Chunks
        strBody = strBody.concat(strChunks);

        byBody = strBody.getBytes();

        return byBody;
}

public byte[] LeaveMessage(String strPeerID,String strSwarmID)
{
        byte[] byBody;
        String strBody,aux;
        char c = 0x00;

        byBody = new byte[36];
        strBody = strPeerID;
        aux = String.valueOf(c);

        while (strBody.length()<20)
                strBody = strBody.concat(aux);

        strBody = strBody + strSwarmID;

        while (strBody.length()<20+16)
                strBody = strBody.concat(aux);

        byBody = strBody.getBytes();

        return byBody;
}

public byte[] FindMessage(String strPeerID,String strSwarmID,String strChunkID)
{
        byte[] byBody;
        String strBody,aux;
        char c = 0x00;
```

```
                byBody = new byte[40];
                strBody = strPeerID;
                aux = String.valueOf(c);

                while (strBody.length()<20)
                        strBody = strBody.concat(aux);

                strBody = strBody + strSwarmID;

                while (strBody.length()<20+16)
                        strBody = strBody.concat(aux);

                strBody = strBody + strChunkID;

                while (strBody.length()<20+16+4)
                        strBody = strBody.concat(aux);

                byBody = strBody.getBytes();

                return byBody;
}

        public byte[] KeepAliveMessage (String strPeerID)
        {
                byte[] byBody;
                String strBody,aux;
                char c = 0x00;

                byBody = new byte[20];
                strBody = strPeerID;
                aux = String.valueOf(c);

                while (strBody.length()<20)
                        strBody = strBody.concat(aux);

                byBody = strBody.getBytes();

                return byBody;
        }

        public byte[] StatReportMessage (String strPeerID,int nNumStats,int nLength,boolean[] boStats,int[]
nStatsLength,String[] strStats)
        {
                byte[] byBody;
                int i,p=0;
                String strBody,aux;
                char c = 0x00;

                byBody = new byte[nLength];
                strBody = strPeerID;
                aux = String.valueOf(c);

                while (strBody.length()<20)
                        strBody = strBody.concat(aux);

                for (i=0;i<20;i++){
                        byBody[i] = (byte)strBody.charAt(i);
                        p++;
                }

                //num stats
                byBody[p]=(byte)(0xff & (nNumStats>>8));        p++;
                byBody[p]=(byte)(0xff & (nNumStats));                   p++;

                //reserved
                byBody[p]=(byte)c;                      p++;
                byBody[p]=(byte)c;                      p++;
```

```
		for (i=0;i<11;i++)
		{
			if (boStats[i]==true)
			{
				byBody[p] = (byte)c; p++;
				switch (i){
					case 0: c=0x00; break;
					case 1: c=0x01; break;
					case 2: c=0x02; break;
					case 3: c=0x03; break;
					case 4: c=0x04; break;
					case 5: c=0x05; break;
					case 6: c=0x06; break;
					case 7: c=0x07; break;
					case 8: c=0x08; break;
					case 9: c=0x09; break;
					case 10: c=0x0A; break;
				}
				byBody[p] = (byte)c; p++; c=0x00;

				byBody[p]=(byte)(0xff & (nStatsLength[i]>>8));p++;
				byBody[p]=(byte)(0xff & (nStatsLength[i]));		p++;

				aux = strStats[i];
				while (aux.length()<(nStatsLength[i]/8))
					aux = aux.concat(String.valueOf(c));

				for (int j=0;j<(nStatsLength[i]/8);j++){
					byBody[p]=(byte) aux.charAt(j);
					p++;
				}
			}
		}
		return byBody;
}

public byte[] StatQueryMessage(int nNumPTypes,boolean[] boStats)
{
		byte[] byBody;
		String strBody,aux;
		char c = 0x00;
		boolean odd = false;

		byBody = new byte[20];
		strBody = "";
		aux = String.valueOf(c);

		//Number of PTypes
		strBody = strBody.concat(String.valueOf((char)(0xff & (nNumPTypes>>8))));
		strBody = strBody.concat(String.valueOf((char)(0xff & nNumPTypes)));

		//Reserved
		strBody = strBody.concat(aux);
		strBody = strBody.concat(aux);

		for (int i=0;i<11;i++)
		{
			if (boStats[i]==true)
			{
				strBody = strBody.concat(aux);
				switch (i){
					case 0: c=0x00; break;
					case 1: c=0x01; break;
					case 2: c=0x02; break;
					case 3: c=0x03; break;
					case 4: c=0x04; break;
```

```
                                        case 5: c=0x05; break;
                                        case 6: c=0x06; break;
                                        case 7: c=0x07; break;
                                        case 8: c=0x08; break;
                                        case 9: c=0x09; break;
                                        case 10: c=0x0A; break;
                                }
                                strBody = strBody.concat(String.valueOf(c));
                                if (odd == true) odd = false;
                                else odd = true;
                        }
                }

                //Reserved for odd
                if (odd == true){
                        strBody = strBody.concat(aux);
                        strBody = strBody.concat(aux);
                }

                byBody = strBody.getBytes();

                return byBody;
        }

        /*******************************/
        //Response Messages

        public byte[] GetMethod(String strMethod){

                char c;
                char[] cAux = new char[4];
                byte[] cMet = new byte[2];

                for (int i=0;i<4;i++)
        {
                c = strMethod.charAt(i+2);
                switch (c)
                {
                        case '0': cAux[i]=0x00; break;
                        case '1': cAux[i]=0x01; break;
                        case '2': cAux[i]=0x02; break;
                        case '3': cAux[i]=0x03; break;
                        case '4': cAux[i]=0x04; break;
                        case '5': cAux[i]=0x05; break;
                        case '6': cAux[i]=0x06; break;
                        case '7': cAux[i]=0x07; break;
                        case '8': cAux[i]=0x08; break;
                        case '9': cAux[i]=0x09; break;
                        case 'A': cAux[i]=0x0A; break;
                        case 'B': cAux[i]=0x0B; break;
                        case 'C': cAux[i]=0x0C; break;
                        case 'D': cAux[i]=0x0D; break;
                        case 'E': cAux[i]=0x0E; break;
                        case 'F': cAux[i]=0x0F; break;
                }
        }

                cMet[0] = (byte) ((0xf0 & (cAux[0]<<4)) | (0x0f & cAux[1]));
                cMet[1] = (byte) ((0xf0 & (cAux[2]<<4)) | (0x0f & cAux[3]));
                return cMet;
        }
}
```