Pixtream: Sistema de media streaming basado en redes P2P

Manuel Alejandro Cerón Estrada

Director: Dr. Pablo Augusto Magé Imbachí

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas
Linea de Investigación en Ingeniería de la Colaboración - Grupo IDIS
Popayán, Mayo de 2010

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Problema

El uso de contenidos multimedia en Internet ha aumentado considerablemente desde que las conexiones de banda ancha empezaron a hacerse populares en los hogares [1]. Una de las aplicaciones multimedia más populares es el *media streaming*. Esta tecnología permite a un cliente recibir audio y vídeo sin necesidad a esperar a que la totalidad del contenido sea descargado o incluso producido [2]. De esta forma, es posible, por ejemplo, empezar a ver un contenido generado en vivo sin necesidad de esperar a que todo sea grabado previamente, o empezar a ver una película inmediatamente y descargarla a medida que se ve.

Los sistemas de *media streaming* tradicionales utilizan una arquitectura cliente/servidor basada en conexiones *unicast*. En un sistema *unicast*, cada cliente tiene una conexión directa con el servidor. Esto tiene la ventaja de que es fácil de implementar y genera relativamente poca latencia[1]. No obstante, existe una gran desventaja: debido a que se tiene que generar una conexión diferente por cada cliente, el ancho de banda de salida requerido para una sesión de *streaming* crece linealmente con el número de clientes [3]. Esto se traduce en unos costos muy altos a la hora de transmitir contenido a una audiencia grande.

El problema general que aborda este proyecto es el de cómo reducir el ancho de banda de salida necesario en un servidor de *streaming*.

1.2. Antecedentes

A principios de los años noventa, el problema del alto requerimiento de ancho de banda en múltiples conexiones *unicast* que transmiten el mismo contenido fue identificado [4]. Una solución a este problema fue agregada al protocolo de Internet: *IP multicasting*. Gracias a esta tecnología es posible enviar paquetes con múltiples destinatarios a través de Internet. Esta solución fue implementada

[1] En este caso se entiende latencia como el tiempo que pasa mientras que el contenido multimedia es producido, hasta que es reproducido por el usuario.

en la capa de red de la pila de protocolos que operan en Internet, siendo los *routers* de Internet los encargados de operar *IP multicasting*. Desafortunadamente, no todos los *routers* que conforman la enorme Internet tienen soporte para *IP Multicasting*, lo cual hace muy difícil utilizar esta técnica para transmitir

Con la aparición de las redes *peer-to-peer* (P2P) a finales de los noventa, se mostró que se podían desarrollar aplicaciones en una forma diferente del clásico modelo cliente/servidor, creando aplicaciones que actúan como clientes y servidores al mismo tiempo y que permiten la formación de redes superpuestas a la red física utilizada [5]. Gracias a las redes P2P se puede pensar en nuevas soluciones al problema del uso del ancho de banda en los sistemas de *media streaming*.

Durante la última década, varias propuestas de sistemas de *media streaming* se han hecho. Varios autores las han analizado y clasificado de acuerdo con la topología que toma la red superpuesta: Topología de árbol, topología de bosque y topología de malla [6].

La mayoría de estas propuestas sólo han sido planteadas de manera teórica y no tienen una implementación real. Los resultados de las propuestas han sido evaluados utilizando simuladores de red, no en un entorno real.

1.3. Descripción del trabajo y objetivos

El objetivo de este proyecto es desarrollar un sistema de *media streaming* basado en redes P2P. Para lograrlo, se estudiaron las propuestas que ya han sido presentadas en otros trabajos, y se analizaron las características en común, seleccionando aquellas que sean más adecuadas para una implementación. Luego de desarrolló un prototipo con las características seleccionadas, se realizaron pruebas de desempeño y una comparación con los resultados obtenidos en simulaciones en los trabajos previos estudiados.

Primero se presenta un marco conceptual del proyecto en el capítulo 2; describiendo la evolución de las soluciones al problema desde las redes *peer-to-peer* y se listan algunas de las principales propuestas plateadas hasta el momento.

El capítulo 3 de este documento presenta el estudio de las propuestas de *media streaming* P2P que se seleccionaron. Cada propuesta es analizada y se determinan sus características más importantes. Luego se comparan las propuestas y sus características para posteriormente realizar una selección de aquellas que deben ser implementadas en el prototipo.

En el capítulo 4 se definen los requisitos del sistema de *media streaming* P2P que se va a implementar y se plantea una arquitectura y un diseño del prototipo. Posteriormente se definen los detalles técnicos y la especificación del protocolo del sistema en el capítulo 5. El capítulo 6 describe el proceso de implementación y pruebas del prototipo a desarrollar.

En el capítulo 7 se explican las pruebas de rendimiento realizadas al prototipo y sus resultados. Posteriormente se realiza una comparación de los datos obtenidos con los resultados teóricos de las propuestas analizadas anteriormente.

Finalmente, en el capítulo 8, se presentan las dificultades que se presentaron

durante la elaboración del proyecto, las propuestas para trabajos futuros y las conclusiones obtenidas.

Capítulo 2

Marco conceptual

2.1. *Media streaming*

El término *media streaming* se refiere a los productos y técnicas utilizados para la transmisión de contenidos multimedia tales como audio y vídeo a través de una red. El nombre se refiere al método de envío más que al medio como tal [2]. El método se caracteriza por la posibilidad de reproducir los contenidos en el cliente sin la necesidad de descargar la totalidad del contenido o incluso mientras este se está generando en tiempo real, por ejemplo usando una cámara web. Esto es posible gracias a que una vez iniciada la transferencia, el programa cliente puede empezar a desplegar contenido al usuario y al mismo tiempo continuar recibiendo la corriente de datos (*stream*) [7].

2.2. El ancho de banda como cuello de botella

El contenido multimedia, en especial el vídeo, suele necesitar muchos *bytes* para ser representado. Es por esta razón que el principal problema para el *media streaming* es al ancho de banda requerido para poder realizar la transmisión. Esto es especialmente importante si se tiene en cuenta que en la mayoría de los casos, no es uno sino muchos los receptores de contenido en un sistema de streaming. Este proceso se conoce como *broadcasting* [7]. Es decir la transmisión de contenido desde una sólo fuente hasta muchos receptores. Dado que el ancho de banda suele ser un recurso limitado y costoso, puede decirse que es un factor determinante para el *media streaming*, toda vez que define el número de clientes hacia los cuales se puede transmitir el contenido.

La arquitectura de *streaming* más simple es la de múltiples conexiones *unicast*. Una transferencia en red tradicional es llamada *unicast*. Se realiza de un computador a otro. A la hora de realizar *streaming*, esto significa que, para lograr transmitir, el emisor de contenidos deberá establecer una conexión *unicast* con cada cliente receptor. El uso de múltiples conexiones implica que el ancho de banda necesario para poder transmitir es equivalente al ancho de banda reque-

rido por una conexión *unicast* multiplicado por todos los clientes que se desea servir. Debido a esto se genera un cuello de botella en cuanto al ancho de banda que se requiere en el lado del servidor de contenidos. Configurar un servidor de media *streaming* con esta tecnología puede resultar muy costoso si se tienen muchos clientes.

Pese al ineficiente uso del ancho de banda, la arquitectura de múltiples conexiones *unicast*, tiene varias ventajas. En primer lugar, la latencia es baja debido a la conexión directa entre el proveedor de contenidos y el receptor. En segundo lugar, dado que la conexión es unidireccional, el programa cliente no necesita establecer conexiones entrantes. En otras palabras, no necesita abrir puertos de comunicación. Esto facilita en gran medida la implantación y distribución en ambientes restringidos. Por ejemplo en lugares en los que se usen cortafuegos o *proxys*, o donde las aplicaciones corren en una "caja de arena" que, por razones de seguridad, les impida realizar acciones como establecer conexiones entrantes. Un ejemplo muy claro sería dentro de un navegador Web. Es por esta razón que esta arquitectura es muy popular hoy en día en el mundo real. Varios programas ampliamente utilizados como IceCast [1] utilizan esta arquitectura. También es la única usada en la web, por sitos de *streaming* como YouTube[2].

2.2.1. Streaming por *Multicast*

Al establecer múltiples conexiones *unicast* se está redundando en cuanto a la transmisión de datos. Si bien se generan muchas conexiones de transmisión diferentes, todas transmiten exactamente el mismo contenido. Los mismos paquetes de información se envían una y otra vez con la única diferencia del destinatario. Para resolver este problema, en 1993 se creo una técnica denominada *multicasting* [8].

Multicasting o multidifusión es una técnica para enviar paquetes de datos desde un punto hacia múltiples destinatarios usando el algoritmo más eficiente posible. Es implementado en la capa tres del modelo OSI [3], a través de los enrutadores. Gracias al *multicasting*, la redundancia se elimina haciendo qué los paquetes de datos se dupliquen sólo cuando encuentran una bifurcación en el camino que llevan hacia sus respectivos destinatarios.

El uso de *multicasting* hace al proceso de media streaming bastante eficiente en cuanto a ancho de banda, dado que sin importar el número de clientes, el ancho de banda requerido siempre será el mismo. No obstante, existen algunos inconvenientes. El principal de ellos es que para poder implementar *multicasting*, se requiere actualizar los enrutadores utilizados en Internet. Si bien la tecnología ya lleva un tiempo considerable, todavía existen muchos enrutadores sin soporte adecuado para *multicast*, lo que hace prácticamente imposible determinar con seguridad si una transmisión *multicast* va a ser posible [8].

[1] `http://www.icecast.org`
[2] `http://www.youtube.com`
[3] Open Systems Interconnection

2.2.2. Redes *Peer-to-Peer*

El concepto *peer-to-peer* o P2P como se suele abreviar, se refiere a una arquitectura de red distribuida en la cual todos los participates de la red actúan como clientes y servidores al mismo tiempo. Compartiendo recursos como poder de procesamiento, ancho de banda o almacenamiento en disco. En una red P2P, los clientes son consumidores y proveedores de recursos, contrastando con la arquitectura cliente/servidor en la que sólo los servidores proveen contenido mientras los clientes consumen [5].

2.2.3. Multicast a nivel de aplicación (ALM)

Con la llegada de las redes superpuestas *peer-to-peer* se demostró el gran potencial que tiene el desarrollar protocolos de red sobre la capa de aplicación. Los programas para compartir archivos basados en redes *peer-to-peer* como Bittorrent, demostraron que es posible acabar con el cuello de botella del ancho de banda en el lado del servidor utilizando inteligentemente el ancho de banda de salida de cada nodo o *peer* en la red [9].

Gracias a la arquitectura P2P, es posible construir un árbol *multicast* que funcione en la capa de aplicación (ALM). El ALM utiliza un sistema *peer-to-peer* para distribuir la transmisión de datos entre todos los nodos de la red. A diferencia del *multicast* tradicional, no son los enrutadores, sino los computadores finales (*peers*) los que se encargan de realizar la replicación de los datos.

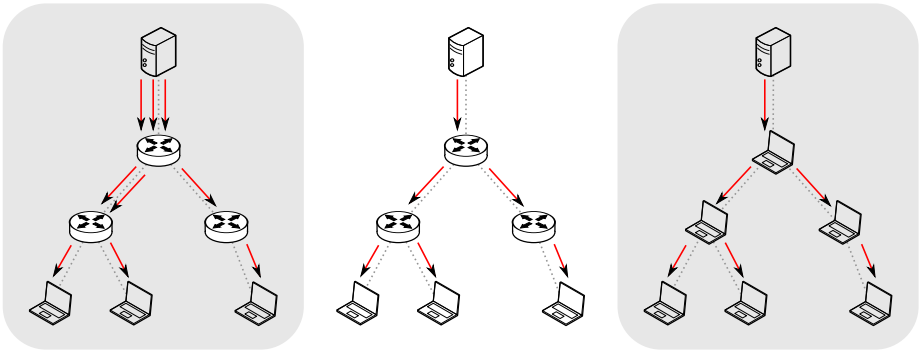


Figura 2.1: Unicasting, IP Multicasting y ALM

La figura 2.1 ilustra las diferencias entre las transmisiones con múltiples conexiones *unicast*, *IP multicasting* y *multicasting* a nivel de aplicación. Las flechas rojas indican las conexiones que se tienen que generar. En el primer caso el servidor tiene que establecer tres conexiones para servir tres cliente. Esto significa que se genera un cuello de botella en el lado del servidor. En el segundo caso (*IP Multicasting*), los datos son dividios en cada enrutador y por lo tanto sólo se tiene que generar una conexión del lado del servidor. Sin embargo, para

realizar la división es necesario contar con enrutadores especiales que no siempre están disponibles. En el tercer caso (ALM), se realiza la división del material en la aplicación. De esta forma no son necesarios enrutadores especiales.

2.3. Clasificaciones de sistemas de *media streaming* basados en redes P2P

El desarrollo de sistemas de *media streaming* basados en redes *peer-to-peer* aún es nuevo. Sin embargo, varios autores han analizado las propuestas realizadas hasta el momento y las han clasificado en distintas categorías. Por lo general, la clasificación se hace basándose en la topología sobre la cual es construida la red superpuesta *peer-to-peer*.

Wen *et al*, consideran que los sistemas de *media streaming peer-to-peer* se clasifican en tres categorías: Basados en topologías de árbol, bosque y malla [6].

En los sistemas que utilizan una topología basada en árbol, los nodos se organizan formando un árbol *multicast* sencillo en el cual la fuente primaria del contenido hace de raíz. Cuando un nuevo nodo ingresa a la red, intenta convertirse en el hijo de la raíz. Si esta no puede tener más hijos debido a limitaciones en el ancho de banda, redireccionará el nodo hacia uno de sus hijos actuales, el cual repetirá el mismo proceso.

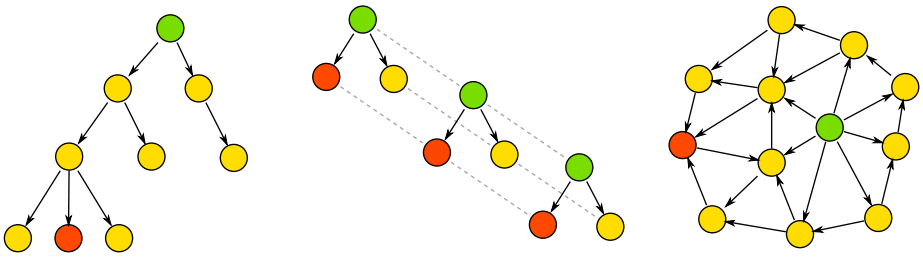
La topología basada en bosque utiliza un proceso similar, pero en lugar de utilizar un sólo árbol, utiliza muchos. Cada nodo que se une a la red debe hacer parte de todos los árboles, siendo un nodo interior en unos y un nodo hoja en otros. El objetivo de esto es optimizar el uso de ancho de banda de salida de todos los nodos, evitando que hayan nodos hoja absolutos.

En la topología basada en Malla, en cambio, no se utiliza una jerarquía organizada. Cada nodo tiene independientemente una lista de nodos receptores, a los cuales le envía paquetes y una lista de nodos emisores, de los cuales recibe paquetes. La red toma entonces una forma lógica parecida a una malla o una telaraña.

Magharei *et al* inicialmente plantean una clasificación esencialmente igual: Topología basada en árbol, topología basada en múltiples árboles y topología basada en malla [10]. Sin embargo, en trabajos más recientes, la distinción entre las categorías de árbol y múltiples árboles es borrada debido a sus similitudes y se plantean sólo dos topologías importantes: Árbol y malla [11]. El sistema de malla también es llamado de enjambre o *swarm* [12].

Carra *et al*, analizan varias propuestas y las clasifican en base a la estructura que usan para despachar el contenido. Estando aquellas en la que los nodos descargan paquetes de un único padre y aquellas que usan un conjunto de padres de los cuales descargan contenido y que son actualizados dinámicamente. Adicionalmente, plantean una nueva categoría: híbrido entre las dos y que formaría un flujo de datos basado en múltiples árboles o una malla [13]. Un enfoque muy similar usan Pianese *et al* describiendo las categorías como redes estructuradas, no estructuradas y de otros tipos [14].

Finalmente, Marfia *et al* realizan una clasificación simple basada en si se usa o no una estructura jerárquica. Agrupan los sistemas en dos categorías: Árbol y malla [15].



Topología de Árbol | Topología de Múltiple Árbol | Topología de Malla

Figura 2.2: Topologías de los sistemas de *media streaming* P2P

2.3.1. Propuestas basadas en topología de árbol

PeerCast

PeerCast [16] es uno de los primero proyectos creados para desarrollar un sistema de *media streaming* basado en *peer-to-peer* y *multicast* a nivel de la capa de aplicación. La topología usada es la de árbol.

ALMI y OverCast

ALMI, por Pendarakis *et al* [4], define e implementa una infraestructura para *multicast* en capa de aplicación. Más que un sistema de *media streaming*, plantea una base sobre la cual se pueden desarrollar sistemas que usen *multicast* simple en capa de aplicación como PeerCast. De manera muy similar funciona OverCast por Jannotti et al [6].

ZigZag

ZigZag, por Tran *et al* [17], presenta una técnica para realizar *media streaming* con una sola fuente de datos. Al igual que PeerCast, usa una topología basada en árbol. ZigZag organiza un árbol de altura fija $O(logN)$ donde N es el número de clientes.

NICE

NICE, por Barnerjee *et al* [18], utiliza un árbol de datos bastante similar a ALMI y Peercast. Se caracteriza por construir una jerarquía de nodos de forma que cada uno lleve información detallada de sus nodos cercanos en la estructura, evitando los lejanos, lo que permite una menor latencia a la hora de transmitir la información.

2.3.2. Propuestas basadas en topología de bosque

Splitstream

SplitStream, por Castro *et al* [19], plantea una técnica para *media streaming* sobre redes *peer-to-peer* que va más allá que aquellas simples basadas en árboles como PeerCast of ZigZag. En lugar, plantea un esquema de bosque o múltiples árboles. Los nodos se agrupan de tal forma que cada uno esté en todos los árboles del sistema, evitando la existencia de nodos hoja absolutos.

Narada

Narada, por Chu *et al* [20], busca desarrollar un protocolo para lograr *multicast* a nivel de la capa de aplicación. Narada es la base para ESM (End System Multicast) [21], un proyecto de investigación iniciado en el área de ciencias de la computación en la Universidad Carnegie Mellon. Posteriormente el equipo de investigadores fundaron una empresa llamada Rinera Networks[4] y hoy desarrollan este sistema comercialmente a través de su producto Conviva.

PALS

PALS, por Magharei *et al* [22], plantea un sistema de media streaming basado en múltiples árboles que sea adaptativo en cuanto al ancho de banda. En PALS se realizan periódicamente adaptaciones cualitativas basadas en el ancho de banda de los nodos emisores de manera que pueda determinar el subconjunto de paquetes que pueden ser enviados hacia los receptores de forma que el ancho de banda se utilice de manera óptima.

CoopNet y Resilient

CoopNet y Resilient por Padmanabhan *et al* [23],[24], utilizan un sistema de múltiples árboles en el cual se introduce redundancia de los datos enviados para lograr robustez y garantizar el envío de todos los paquetes.

2.3.3. Propuestas basadas en topología de malla

PULSE

PULSE, por Pianese *et al* [14], plantea la construcción de un sistema no estructurado, basado en malla y dirigido entorno al flujo de datos para realizar *media streaming*.

2.3.4. DONet/CoolStreaming

DONet/CoolStreaming por Zhang emphet al [25], es un sistema basado en malla que no sólo se ha quedado en propuesta sino que ha sido implementado en Roxbeam, un servicio comercial de *media streaming* en China [6]. DO-

[4] `http://www.rinera.com`

Net/Coolstream probablemente sea el sistema de *media streaming peer-to-peer* más exitoso construido hasta el momento [15].

2.3.5. BEAM

PRIME por Magharei *et al* [12], propone un modelo para lograr efectivamente un sistema de enjambre o *swarming*. Identifica y pretende resolver dos problemas fundamentales de *media streaming*: El cuello de botella en el ancho de banda y el cuello de botella en el contenido. Se basa en el trabajo de PALS. Por otro lado BEAM por Purandare *et al* [26], es otro sistema que busca generar un enjambre. BEAM busca agrupar los nodos en pequeños grupos llamados alianzas, los cuales se comunican para compartir partes de un paquete de contenido.

2.3.6. Bittorrent y GnuStream

GnuStream, por Jiang *et al* [27], presenta una adaptación del sistema de transferencia de archivos Gnutella [28] para el caso de *media streaming*. De la misma manera, Shah et al [9] plantean una adaptación del protocolo Bittorent [5] para este propósito.

2.3.7. AnySee

AnySee por Xiaofei *et al* [29], presenta un sistema de *streaming peer-to-peer* enfocado en disminuir la latencia y garantizar la calidad del contenido transmitido.

2.3.8. *Media streaming P2P* en la industria

China es uno de los principales lugares donde la industria del *media streaming* basado en redes *peer-to-peer* ha florecido. Desde el momento en que aparecen las primeras implementaciones prácticas de estos sistemas, surge una industria al rededor en China [6].

Varias empresas han empezado a distribuir contenidos a través de este tipo de redes. Por ejemplo: PPLive[6], PPStream[7], Mysee[8], Roxbeam[9] y UUSee[10].

Estas implementaciones son privativas y por lo tanto difíciles de analizar para saber cómo funcionan. Sin embargo, algunos estudios han realizado ingeniería inversa para determinar la arquitectura en la que se basan. En el caso de Roxbeam, por ejemplo, se ha determinado que está basado en CoolStreaming de Marfia *et al* [15].

[5] `http://www.bittorrent.com`
[6] `http://www.pplive.com/en`
[7] `http://www.ppstream.com`
[8] `http://www.mysee.com`
[9] `http://www.roxbeam.com`
[10] `http://www.uusee.com`

2.3.9. Análisis de sistemas de *media streaming* basados en redes P2P

Varios estudios se han dedicado a analizar las diferentes propuestas de sistemas de *media streaming* basados en redes *peer-to-peer*.

Magharei *et al* comparan los sistemas basados en topología de árbol y de malla y, mediante el uso de simulaciones, miden el rendimiento de ambas. Sus resultados indican que los sistemas basados en malla exhiben consistentemente un rendimiento mejor a aquellos basados en árbol. [11]

Marfia *et al* realizan experimentos sobre algunas implementaciones existentes de sistemas de *media streaming* basados en redes *peer-to-peer*. Concluyen que los sistemas híbridos como CoopNet y AnySee presentan un mejor rendimiento [15].

La figura 2.3 ilustra la relación entre las distintas propuestas de sistemas de *media streaming* basados en redes *peer-to-peer* y las topologías en las cuales se basan. Es fácil ver una tendencia desde topologías simples en árbol hasta las más complejas en malla. También es fácil observar cómo las investigaciones actuales se basan casi completamente en topologías de malla.

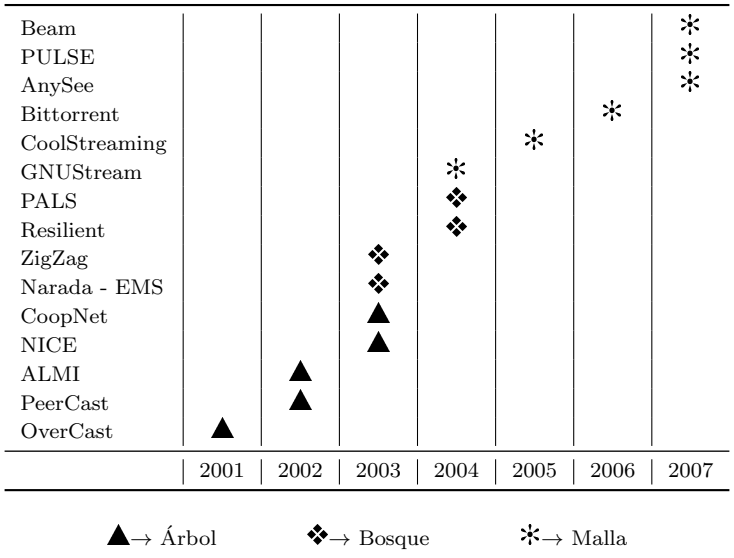


Figura 2.3: Línea de tiempo de las propuestas de *streaming* P2P

Capítulo 3

Estudio de las propuestas de *media streaming* P2P

La primera parte del proyecto Pixtream consiste en estudiar algunos de los trabajos previos más relevantes sobre el tema de *media streaming*. Para esto se seleccionaron seis propuestas: Bittorrent para *streaming*, GNUStream, BEAM, AnySee, PULSE y DONet/Coolstreaming. Durante el estudio de cada una de las propuestas se seleccionaron las características más importantes de cada una, las cuales son posteriormente comparadas y utilizadas para el diseño de Pixtream.

3.1. BitTorrent

BitTorrent es un protocolo de intercambio de archivos basado en redes P2P. Fue desarrollado por Bram Cohen en 2001 [30]. BitTorrent plantea un esquema de enjambre o *swarming* para realizar efectivamente las transferencias de archivos [31]. El objetivo principal es optimizar el uso del ancho de banda de todos los pares en la red. Actualmente el protocolo BitTorrent es responsable del uso de más de 30 % del ancho de banda de toda Internet [32].

El protocolo BitTorrent divide el archivo que se pretende compartir en varios pedazos de tamaño similar. Para poder comprobar en todo momento la integridad de las partes, por cada pedazo se crea un *hash* criptográfico utilizando el algoritmo SHA1 [33].

Existen tres tipos de participantes en una red BitTorrent: Los rastreadores o *trackers*, los pares semillas o *seeds*, y los pares consumidores o *leechers*. El rastreador es el componente encargado de mantener la lista de todos los pares de la red y de hacer que los pares se puedan contactar entre sí. Se encarga del proceso de ingreso a la red o *bootstrapping*. Cuando un nuevo cliente quiere unirse a la red para descargar un archivo, primero contacta al rastreador, quien le entrega una lista de todos los pares conectados actualmente descargando el archivo deseado. Con esta lista, los clientes pueden iniciar individualmente conexiones con otros y solicitar pedazos del archivo que se está descargando. Los

pares semillas son aquellos que tienen la totalidad del archivo y están enviando partes de este a pares consumidores. Los pares consumidores son aquellos que sólo tienen parte o nada del archivo.

Cada nodo de una red BitTorrent utiliza varios algoritmos para decidir a qué pares enviar datos y de qué pares recibir. Los objetivos de estos algoritmos son evitar cuellos de botella en la disponibilidad de paquetes e incentivar, en la mayor medida, la contribución de ancho de banda de salida de todos los nodos participantes. Para lograr el primer objetivo, BitTorrent utiliza un algoritmo denominado "El más raro primero" o *"Rarest first"*. Cuando un par desea pedir un pedazo en especial, se decide por aquel paquete más raro en la red, es decir, aquel que sea poseído por menos pares. De esta forma, se evitan los paquetes raros al máximo y por tanto se evita la situación en la que que múltiples pares tengan que luchar por un mismo recurso. Para cumplir el segundo objetivo, se utiliza un algoritmo llamado "Unas por otras" *"Tit for tat"*. Utilizando este algoritmo, un par decide a qué pares enviar paquetes prefiriendo aquellos que le hayan proporcionado datos recientemente. Esto evita el comportamiento egoísta de algunos pares que se dedican exclusivamente a bajar contenido de la red sin subir paquetes a ningún otro par. Para evitar que los nodos recién ingresados se queden sin recibir datos, los pares mandan un cierto número de sus paquetes a pares aleatorios.

Pese a que BitTorrent es un protocolo muy efectivo para la transferencia de archivos, usarlo para transmitir contenido multimedia en vivo es prácticamente imposible. Esto se debe a que BitTorrent cuenta con el hecho de tener un archivo completo desde el inicio de la descarga para poder dividirlo en pedazos y crear los *hash* criptográficos. El algoritmo *"Rarest first"* necesita una disponibilidad de todas las partes desde el comienzo de la transferencia y el algoritmo *"Tit for tat"* causa que los nodos recién ingresados a la red comiencen la descarga lentamente.

Pese a los inconvenientes, Shah y Pâris creen que el protocolo BitTorrent se podría utilizar para transmitir contenido multimedia en vivo si se le hacen algunas modificaciones [9]. Los autores proponen varios cambios al protocolo para este propósito. En primer lugar, eliminan la necesidad de tener el archivo completo al inicio de la transferencia, introduciendo lo que llaman "ventana deslizante" o *"sliding window"*. La ventana deslizante es un espacio temporal de almacenamiento o *buffer* que contiene los siguientes n pedazos de contenido que el usuario va a ver en un determinado lapso t. A medida que el flujo de datos es transmitido desde la fuente multimedia, la ventana se va corriendo poco a poco. Aquellos paquetes que no pudieron ser obtenidos antes de que la parte izquierda de la ventana llegue hasta ellos deberán ser descartados.

El algoritmo *"Rarest first"* original de BitTorrent aplicaría ahora exclusivamente a la ventana deslizante. Esto limitaría un poco el desempeño general del algoritmo de selección de paquetes para el caso de transferencias de archivos, pero para *media streaming* aumentaría la calidad de la transmisión, donde lo más importante es la llegada a tiempo de los paquetes necesarios.

Para lograr transmitir contenido multimedia en tiempo real usando BitTorrent, es necesario también agilizar el proceso de iniciación de los pares al proceso

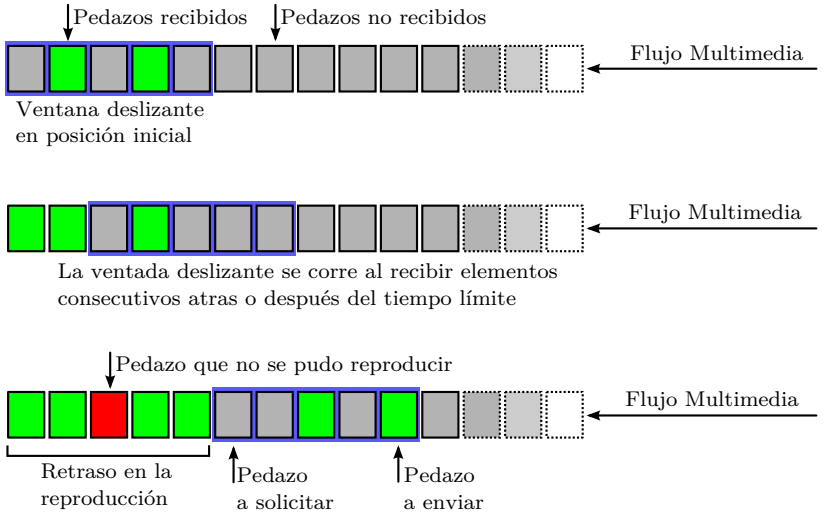


Figura 3.1: Ventana deslizante usando el algoritmo *rarest first* con BitTorrent.

de transferencia. A causa del algoritmo *"Tit for tat"*, esto no es posible con el protocolo tradicional, debido a que los nodos inician con cero pedazos y tienen que esperar a que algunos paquetes les lleguen aleatoriamente. Para sobrellevar esto, Shah y Pâris proponen usar un algoritmo aleatorio de selección de pares al inicio de la transmisión, logrando que el proceso de iniciación sea lo más rápido posible. Una vez que todos los pares tengan partes suficientes, entonces se usará *"Tit for tat"* para controlar el abuso por parte de los usuarios egoístas.

Pruebas en simulaciones realizadas a la propuesta de Shah y Pâris, demuestran que aplicando estas modificaciones al algoritmo BitTorrent, se puede lograr que más del 90 por ciento de los paquetes sean entregados a tiempo para una correcta reproducción de un flujo multimedia.

3.2. Gnustream

Gnustream, por Jiang *et al* [27], es una propuesta de un sistema de *media streaming* P2P basado en el protocolo Gnutella [28].

Gnutella es una red de intercambio de archivos P2P basada en un protocolo con su mismo nombre. El protocolo Gnutella fue diseñado para conseguir una red completamente descentralizada que no dependiera de ningún servidor o control principal. A finales de 2007 la red Gnutella fue el sistema de intercambio de archivos más usado en Internet con más de dos millones de computadores interconectados [32].

La red Gnutella fue creada en 2000 por Justin Frankel y Tom Pepper de la empresa NullSoft, después adquirida por AOL. Surgió como una manera de contrarrestar las debilidades de las redes de intercambios de archivo semi-

centralizadas como Napster y FastTrack (también conocida como Kazaa). El protocolo utilizado por el cliente original fue descubierto utilizando ingeniería inversa y existen varios proyectos libres y propietarios implementándolo y desarrollándolo paralelamente [34].

Gnutella se caracteriza por ser una red P2P pura, es decir, sin ningún tipo de servidor central. En la red existe un sólo tipo de entidad: El par, el cual actúa como cliente y servidor al mismo tiempo. Existen tres operaciones importantes que un par puede realizar en una red Gnutella: El ingreso a la red o *bootstrapping*, la búsqueda de contenido y la descarga de archivos [28].

Para lograr ingresar a la red, un par de Gnutella debe contactar al menos otro par que ya esté en la red. A través de los años se han implementado distintas técnicas para lograr este objetivo. Está fuera del alcance de este documento mencionar todas.

Para la búsqueda, Gnutella usa un algoritmo llamado "Inundación de la red" [35]. Cuando un nodo desea buscar algún contenido, envía un mensaje a todos los nodos a los cuales esté conectado. Estos nodos reciben el mensaje y responden si poseen el contenido, retransmitiendo el mensaje a todos los nodos que a su vez estén conectados a ellos; los cuales repetirán el mismo procedimiento.

La descarga del archivo se hace directamente de todos los nodos que contestaron el llamado de búsqueda. Adicionalmente, Gnutella implementa un sistema de mensajes que permiten rastrear el estado de los pares conectados y determinar rápidamente cuando un par se ha desconectado o cuando ya no está disponible.

El protocolo Gnutella se enfoca en la construcción de la red P2P que fundamenta el intercambio de archivos entre pares. Sin embargo, Gnustream propone una manera de utilizar la misma red P2P en la transmisión de contenido multimedia.

Gnustream propone una red con las siguientes características:

1. Agregación de múltiples transmisores. A diferencia de otras propuestas de *streaming* P2P como Narada [20] y PeerCast [16] que funcionan creando un árbol *multicast* a nivel de aplicación y en la cual cada nodo de la red recibe información de un sólo nodo receptor. Gnustream usa la naturaleza de múltiples transmisores de Gnutella para recibir contenido multimedia desde distintos pares, balanceando la carga de transmisión entre ellos. Para ello utiliza dos políticas diferentes dependiendo de la situación: Distribución uniforme y distribución proporcional. La distribución uniforme se usa en redes con ancho de banda homogéneo como *intranets*. La distribución proporcional se usa en redes heterogéneas y dinámicas como Internet.

2. Re-agrupación de datos recibidos. El receptor obtiene múltiples paquetes en diferente orden y se encarga de re-ordenarlos y concatenarlos formando un flujo continuo de multimedia que luego puede ser enviado a un programa reproductor.

3. Detección de cambios en el estado de los pares. Gnustream usa sondeos constantes a los pares conectados para verificar el estado de estos y reaccionar ante desconexiones o inactividad.

4. Recuperación de falla y degradación. Gnustream usa un algoritmo para detectar degradaciones en el desempeño de los pares transmisores tales como disminución en la tasa de transmisión o desconexión y permite escoger a tiempo fuentes alternativas para un determinado paquete.

3.3. BEAM

BEAM, acrónimo en inglés para *Bit strEAMing*, es otra propuesta de *media streaming* P2P [26]. BEAM no se basa en ningún protocolo P2P existente, sin embargo, por sus componentes, se puede deducir que usa un sistema muy parecido a BitTorrent.

BEAM propone un sistema P2P con tres entidades básicas: Un servidor multimedia, un rastreador o *tracker*, y los pares. El servidor multimedia se conecta a un flujo de datos de audio y/o vídeo y lo divide en diferentes pedazos secuenciados o paquetes que luego son transmitidos por toda la red P2P. Al igual que en el protocolo BitTorrent, el rastreador administra la lista de nodos conectados al sistema y es usado para el proceso de ingreso a la red. Adicionalmente, en BEAM, el rastreador se comunica periódicamente con el servidor multimedia para intercambiar el estado del sistema e información. Los pares reciben paquetes de datos del servidor multimedia y de otros pares y los reconstruyen una vez los tengan en orden secuencial.

Para cumplir con el objetivo de enviar y recibir los paquetes multimedia a tiempo, logrando una reproducción continua, BEAM utiliza dos estrategias:

1. Organizar los nodos en pequeños grupos llamados "alianzas" para el mutuo beneficio lo cual intenta minimizar el gasto de tiempo de reacción cuando varios pares compiten por un sólo paquete.

2. Recompensar los nodos que más contribuyen en ancho de banda de salida, permitiéndoles conectarse directamente al servidor multimedia.

Las "alianzas" son grupos de entre cuatro o seis pares que forman una asociación simbiótica entre ellos. Las alianzas las forman los pares individualmente con aquellos que han intercambiado información. Cada vez que un nodo de una alianza recibe un nuevo paquete, utiliza un mensaje de anuncio a sus nodos aliados informándolo. Esto hace que los otros pares de la alianza eviten descargar ese paquete de nodos externos, ya que posteriormente lo obtendrán de ese aliado. El proceso además disuade a usuarios egoístas ya que aquellos que no compartan paquetes no podrán formar alianzas.

Adicionalmente, BEAM define una métrica denominada "Factor de Utilidad" (UF por sus siglas en inglés) la cual determina el grado de contribución de un par. Aquellos pares que puedan compartir más paquetes debido a que disponen de mucho ancho de banda de salida tiene un UF alto, mientras que aquellos que no pueden compartir adecuadamente tienen un UF bajo. Los nodos actualizan periódicamente su UF al rastreador quien a su vez comunica esos datos al servidor multimedia. El servidor multimedia transmite sus paquetes a los

nodos con UF más alto con el objetivo de garantizar que la distribución sea lo más rápido y eficiente posible. Evitando cuellos de botella causados por nodos débiles recibiendo contenido primario.

3.4. AnySee

AnySee es una propuesta creada por Liao *et al* para un sistema P2P de *streaming* en tiempo real [29]. AnySee se enfoca en tres problemas fundamentales de un sistema de *media streaming* P2P: Retraso de inicio, retraso desde la fuente hasta el cliente final, y la garantía de una reproducción continua.

En sistemas como Gnustream o BEAM, la red toma una forma de malla creada casi aleatoriamente y el flujo multimedia fuente es dividido en pedazos que son luego transmitidos a diferentes nodos. En AnySee se crean distintas redes superpuestas [1] en las cuales el flujo multimedia es transmitido en forma continua por un camino de nodos, desde el nodo fuente hasta el nodo final. Cada red superpuesta tiene una forma más parecida a un árbol *multicast* [4]. Debido a que se forman varias redes superpuestas diferentes, se evitan los problemas típicos de las topologías de árbol, como por ejemplo el hecho de que los nodos hoja no contribuyan ancho de banda de salida.

En AnySee, además, la creación de la red no es hecha al azar sino de forma ordenada. El objetivo máximo de AnySee es lograr la mayor equivalencia posible entre la topología lógica de la red y la topología física subyacente. Es decir, que AnySee busca hacer que los nodos interconectados en la red superpuesta sean, en lo posible los más cercanos físicamente. Además, AnySee también busca realizar optimizaciones entre las distintas redes superpuestas creadas, de tal forma que se pueda encontrar la distancia más corta entre dos nodos incluso aún pasando por distintas redes superpuestas.

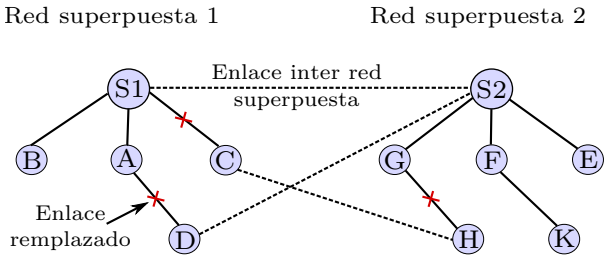


Figura 3.2: Redes superpuestas en AnySee.

Los propósitos concretos de AnySee son:

[1] Una red superpuesta es una red que está construida encima de otra. Los nodos de la red superpuesta pueden estar diseñados como si estuvieran conectados de forma lógica o virtual que puede corresponder a uno o más enlaces físicos.

1. Mejorar la utilización global de recursos de la red de *streaming* P2P distribuyendo el tráfico a través de todos los enlaces físicos de forma pareja.

2. Asignar recursos basándose en la distancia física y el retraso de transmisión.

3. Garantizar la calidad del servicio utilizando los pares más cercanos para la transmisión, así estos pares se encuentren en redes superpuestas diferentes.

4. Balancear la carga a través del grupo de miembros.

AnySee usa un algoritmo de detección de ubicación llamado LTM (*Location Topology Matching*) [36]. El cual utiliza estampas de tiempo para calcular cuanto tarda un mensaje en pasar de un nodo a otro. Para que esto sea posible, todos los nodos utilizan un reloj sincronizado utilizando el protocolo NTP (*Network Time Protocol*) [37].

AnySee utiliza dos sistemas diferentes de optimización: Optimización de red superpuesta sencilla y optimización inter-red superpuesta. Debido a que AnySee forma redes superpuestas en forma de árbol, utiliza esquemas parecidos a otras propuestas de P2P *streaming* como Narada [20]y DONet [25], para manejar los posibles problemas que se puedan presentar en el árbol como el retiro de nodos o la congestión de caminos y en general el manejo de ingreso y egreso de nodos en la red.

El administrador de optimizaciones entre redes superpuestas utiliza un sistema de enlaces de respaldo para un determinado camino de *streaming*. Por cada nodo en la red, existe un camino activo de nodos desde la fuente de datos hasta el par. Además del camino activo, existen también varios caminos alternativos que no se usan, pero se tienen guardados en caso de ser necesitados. Si por alguna razón el camino activo se rompe o su desempeño es pobre, el sistema de optimización cambia automáticamente de camino.

3.5. PULSE

PULSE por Pianese *et al* [14], es una de las primeras propuestas de *media streaming* P2P que se enfoca exclusivamente a escenarios en donde los nodos tienen un rango heterogéneo y variable de recursos y ancho de banda.

PULSE busca recompensar la participación de los pares y desincentivar aquellos que comparten una cantidad insuficiente de recursos. PULSE encara estos objetivos introduciendo varios tipos de incentivos a la cooperación en una red P2P no estructurada y orientada a los datos. Para los proponentes de este sistema, una red no estructurada es aquella que no se forma a partir de una jerarquía establecida, su topología no toma ninguna forma específica, se trata más bien de que cada par establezca conexiones a su discreción con otros pares, formando algo parecido a una malla.

Para mejorar el desempeño del sistema, PULSE, al igual que otros sistemas como BEAM, plantea que es importante situar los nodos de la red de acuerdo con

su rendimiento individual de intercambio. Esto implica que los nodos con mayor capacidad de distribución deben estar conectados lo más directamente posible al flujo de datos, mientras que aquellos pares con recursos escasos deberían estar alejados, ya que podrían repercutir en el desempeño general de la transmisión.

PULSE aboga por un control individual de los nodos en el proceso de formación de la red. Cada par es libre de deambular por la red, conectándose con los nodos que considere necesarios y reaccionar a cambios globales y variaciones de la capacidad local de transmisión a través del tiempo.

PULSE no propone detalles sobre la red P2P subyacente y deja a los implementadores decisiones sobre la construcción de esta. Propone que los anuncios sobre los cambios globales y *bootstrapping* puedan ser realizados utilizando un sistema de rastreador o *tracker* como en BitTorrent, o podría usarse un protocolo de mensajería como GOSSIP [38] al igual que se hace en Gnutella. Independientemente de la red P2P subyacente, PULSE propone dos componentes fundamentales: la fuente de *streaming* y los nodos.

La fuente de *streaming* se encarga de codificar y dividir el flujo multimedia en una serie de pedazos. Los pedazos son numerados secuencialmente y etiquetados con una marca de tiempo.

Los nodos de PULSE son funcionalmente idénticos. Son libres de intercambiar información de control y pedazos de datos. Las asociaciones entre los nodos son el resultado de decisiones independientes. Las conexiones son independientemente establecidas por los pares desde una red de enlaces de control e intercambio de datos. Los nodos escogen a qué pares conectarse basándose en distintos parámetros. Los más importantes son las características topológicas de la red subyacente, los recursos disponibles en cada par y los resultados de los propios algoritmos de selección y obtención de cada par local.

La estructura de un par de un sistema PULSE cuenta con los siguientes componentes:

1. El "*buffer*" de datos, donde los pedazos son almacenados antes de reproducirse.

2. El "registro de conocimiento" donde se guarda información acerca de la presencia, contenido de datos, relaciones pasadas y asociaciones actuales de cada par.

3. La "Lógica de intercambio", cuyo papel es requerir pedazos de los vecinos y escoger y programar aquellos que van a ser enviados.

Al igual que otras propuestas, el *buffer* de datos utiliza un sistema de ventana deslizante para regular la recepción del flujo multimedia. PULSE denomina "borde del *buffer*" al pedazo más a la izquierda de la ventana deslizante y "zona de interés" a los pedazos que van a ser necesitados pronto por un par para poder reproducir el contenido multimedia fluidamente.

En el "registro de conocimiento", PULSE agrupa los pares en dos listas: *"missing"* y *"forward"*. Los pares en *missing* son aquellos cuya ventana deslizante se sobrepone con la propia, es decir, que ambos están solicitando y recibiendo pedazos que se encuentran bajo el mismo rango y por lo tanto pueden

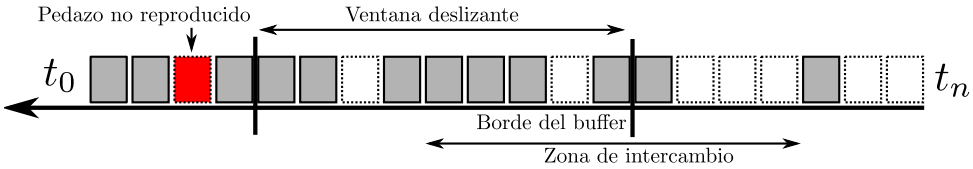


Figura 3.3: Ventana deslizante en PULSE.

beneficiarse mutuamente enviándose los pedazos que el otro necesita. Los pares en *forward* son aquellos que operan sobre una ventana deslizante diferente y por lo tanto no pueden aportar pedazos al par, a estos pares se les envía pedazos de forma altruista.

3.6. CoolStreaming/DONet

DONet [25] es otro sistema de *media streaming* P2P. Plantea una estructura muy simple en la cual cada par de la red establece relaciones con otros e intercambia información y pedazos de datos. Cada par tiene un conjunto de pares proveedores y un conjunto de pares receptores. DONet fue una de las primeras propuestas de *media streaming* P2P con topología de malla real y una estructura simple dirigida por el recibidor y los datos. DONet no sólo se ha quedado en propuesta, sino que ha sido implementado en CoolStreaming, que se ha convertido en uno de los sistemas de *media streaming* P2P más populares de Internet [39]. La observación más importante que se ha hecho sobre Coolstreaming, según sus autores, es que la calidad del *streaming* aumenta a medida que el tamaño de la red crece.

Las características principales que plantea DONet son:

1. Fácil de implementar. DONet propone un sistema con una estructura sencilla. Cada par mantiene conexiones con otros pares a los cuales provee de datos y de los cuales recibe datos. A diferencia de otras propuestas como AnySee (véase sección 3.4) que buscan conformar una estructura de árbol compleja, DONet crea algo parecido a una malla aleatoria.

2. Eficiencia. Debido a que el envío y recepción de datos es determinado dinámicamente de acuerdo con el estado y capacidades actuales de cada par.

3. Robustez y elasticidad. Las asociaciones entre pares permiten cambios rápidos de entre muchos proveedores.

DONet plantea una red P2P con dos elementos básicos. Un par fuente que se conecta directamente a una fuente de *streaming* tradicional, y los pares ordinarios. DONet utiliza un sistema de mensajes GOSSIP [38] para la comunicación

entre los pares. A diferencia de sistemas como BitTorrent que usan un rastreador para el proceso de entrada a la red, DONet requiere la conexión directa con el nodo fuente, quien posteriormente diferirá el proceso de *bootstrapping* a otro nodo en la red. Esto evita la necesidad de utilizar un rastreador o *tracker*.

Cada nodo de DONet tiene un identificador único y mantiene una lista parcial de nodos compañeros llamada *mCache*. Cuando un nodo ingresa a la red, primero contacta el nodo origen o fuente, este inmediatamente lo redirecciona hacia un par aleatorio en la red quien compartirá su *mCache*. Los nodos comparten sus *mCaches* utilizando un protocolo llamado SCAMP (*Scalable Gossip Membership Protocol*) [40].

Al igual que muchas otras propuestas, DONet divide el flujo multimedia en varios pedazos y los distribuye de forma diferente entre todos sus pares. La disponibilidad de las partes de un par y sus compañeros es almacenada en una estructura llamada *Buffer Map*. Para lograr un flujo continuo de datos, se usa un sistema de ventana deslizante o *sliding window* que permite representar los datos necesarios para reproducir n segundos de contenido.

Dados el *Buffer Map* de un nodo y de sus compañeros, se puede generar un plan para solicitar y recibir segmentos de los compañeros. El algoritmo de selección trata de cumplir con dos restricciones: el tiempo que un pedazo requiere para ser reproducido y el ancho de banda del par que lo posee. El algoritmo de selección de DONet funciona de manera similar al algoritmo *Rarest first* de BitTorrent. Primero se calcula el número de potenciales proveedores para un pedazo del contenido, es decir, aquellos pares compañeros que contengan ese pedazo. Luego determina qué pedazo bajar empezando por aquel que contenga el menor número de potenciales proveedores. Si hay múltiples posibles proveedores, se escoge aquel que tenga el mayor ancho de banda disponible y el menor tiempo.

Al igual que BitTorrent, DONet genera un cierto número de peticiones y envío de pedazos aleatorio para evitar cuellos de botella al iniciar la transferencia. Además de eso, los nodos que abandonan la red mandan un mensaje de salida para informar de que abandonan y evitar que otros nodos cuenten con sus pedazos.

3.7. Comparación de características y algoritmos

Al analizar las diferentes propuestas de *media streaming* P2P, es fácil notar que existen varias características que son comunes entre ellas. En esta sección se realiza una comparación de cada planteamiento, que representa la base para la implementación del prototipo *software*.

Se seleccionaron doce características diferentes para realizar la comparación de las propuestas de *streaming* P2P:

1. **Topología de la red superpuesta**. ¿Qué forma lógica toma la red? ¿Árbol, múltiples árboles o malla?

2. **Red P2P subyacente**. ¿Se basa la propuesta en alguna arquitectura P2P existente? ¿Cuál?

3. **Ingreso a la red o *bootstrapping***. ¿Qué estrategia usa el sistema para el proceso de ingreso a la red? ¿Es una red P2P pura[2]? ¿O necesita una entidad centralizada para este proceso?

4. **Esquema de distribución**. ¿Qué esquema se usa para compartir el contenido multimedia de tal forma que a cada par le toque una parte diferente?

5. **Sistema de búsqueda**. ¿El sistema soporta búsqueda de contenido? ¿Cómo se busca y se accede al contenido?

6. **Control de continuidad**. ¿Qué estrategia emplea el sistema para sobrellevar el problema de la no disponibilidad de la totalidad del contenido desde el inicio de la transferencia?

7. **Componentes**: ¿Cuáles son los componentes que conforman el sistema? ¿Se usan varios programas? ¿Cuáles?

8. **Incentivos para la contribución**: ¿De qué forma el sistema incentiva a los pares a que compartan sus recursos, por ejemplo ancho de banda, a la red?

9. **Control de la estructura de la red** ¿Qué entidad es la encargada de controlar quién se conecta con quién? ¿Cómo se controla la topología de la red?

10. **Política de envío de pedazos** ¿Qué algoritmo se usa para decidir a qué par se deben enviar partes del flujo multimedia?

11. **Política de petición de pedazos** ¿Qué algoritmo se usa para decidir de qué par se debe recibir contenido y a qué par solicitar contenido?

12. **Estrategia de asociación entre pares** ¿Qué estrategias de asociación entre pares existen?

En las tablas 3.1 (página 23) y 3.2 (página 24) se puede encontrar una comparación de los trabajos estudiados, de acuerdo a las características mencionadas.

3.8. Selección de características

Basándose en la comparación de sistemas de *media streaming* P2P que se realizó en la sección 3.7, se seleccionaron una serie de características para ser implementadas y posteriormente comparadas con los resultados teóricos planteados por las propuestas.

[2]Una red *peer-to-peer* pura es aquella que no usa ningún servidor central [41].

Propuesta	Topología	Red subyacente	Ingreso a la red	Esquema de distribución	Búsqueda	Control de continuidad	Componentes
Bittorrent:	Malla	Bittorrent	Rastreador	División del flujo multimedia	Archivo *.torrent*	Ventana deslizante	▪ Rastreador ▪ *Seeds* ▪ *Leechers*
GNUStream:	Malla	Gnutella	GOSSIP	División del flujo multimedia	Inundación de la red	No especificado	▪ Pares
BEAM:	Malla	Ninguna	Rastreador	División del flujo multimedia	No especificado	Ventana deslizante	▪ Rastreador ▪ Servidor Multimedia ▪ Pares
AnySee:	Árbol múltiple	Ninguna	Conexión al par fuente	Múltiples redes superpuestas	No especificado	Flujo continuo en árbol *multicast*	▪ Par fuente ▪ Pares receptores
PULSE:	Malla	Ninguna	Rastreador o GOSSIP	División del flujo multimedia	No especificado	Ventana deslizante	▪ Fuente de *streaming* ▪ Pares
DONet:	Malla	Ninguna	Conexión al par fuente	División del flujo multimedia	Protocolo SCAM	Ventana deslizante	▪ Par fuente ▪ Pares regulares

Tabla 3.1: Comparación de sistemas de *media streaming* P2P parte 1

Propuesta	Incentivos a contribución	Control de la estructura	Política de recepción	Política de envío	Asociaciones entre pares
Bittorrent:	Envíos aleatorios al inicio de la transferencia y *"Tit for tat"* después.	Individualmente. Semi-aleatorio.	*Rarest first.*	*"Tit for tat"*	Ninguna.
GNUStream:	Distribución proporcional. Cada par envía datos de acuerdo con su capacidad.	Individualmente. Semi-aleatorio.	Cualquier par que contenga el pedazo requerido y tenga la capacidad de proveerlo.	A los pares que lo requieran hasta cumplir con el tope en ancho de banda	Ninguna.
BEAM:	Pares con mejor *Factor de utilidad* cerca de la fuente de *streaming*.	Individualmente. Semi-aleatorio.	Cualquier par que contenga un pedazo, con prioridad para aquellos que pertenezcan a una alianza.	A los pares que lo requieran con prioridad a aquellos que pertenecen a alianzas.	Los pares se agrupan en alianzas.
AnySee:	No está contemplado dentro de la propuesta.	El par fuente controla varios árboles *multicast*.	Selección de los pares que esten más cerca físicamente usando el algoritmo LTM	Selección de los pares que estén más cerca físicamente usando el algoritmo LTM	Ninguna.
PULSE:	Situar los nodos en la red de acuerdo con su rendimiento individual.	Individualmente por par. Los nodos se sitúan de acuerdo con su rendimiento.	Recepción de cualquier par que contenga el pedazo requerido.	Prioridad para aquellos que se encuentren en la lista *"missing"* y de forma altruista a los de la lista *"forward"*.	Ninguna.
DONet:	No está contemplado. Se compensa con una política de selección con base en el ancho de banda.	Individualmente. Semi-aleatorio.	*Rarest-first.*	Se envían paquetes de acuerdo con las posibilidades de ancho de banda a los pares que lo soliciten.	Ninguna.

Tabla 3.2: Comparación de sistemas de *media streaming* P2P parte 2

Las características han sido seleccionadas de acuerdo con las categorías mencionadas en la sección 3.7. Por cada categoría, se tomaron en cuenta las distintas variaciones que presenta cada propuesta y se escogió una para ser implementada en Pixtream.

A continuación se presenta el análisis por cada categoría mencionada y la selección que se realizó para Pixtream.

3.8.1. Topología de la red superpuesta

La topología se refiere a la forma en la que se organizan los nodos en la red P2P. Desde un principio, este proyecto ha tenido el propósito de analizar únicamente propuestas basadas en una topología de malla. Sin embargo, como se observó anteriormente, algunas propuestas se presentan a sí mismas como de topología de malla, pero en realidad su comportamiento es más cercano al de un arreglo de múltiples árboles. Tal es el caso de AnySee.

Como se puede observar en la tabla 3.1 (página 23), a excepción de AnySee (sección 3.4), todas las propuestas analizadas utilizan topología de malla. Además, durante el estudio del estado del arte de los sistemas de *media streaming* P2P, se logró encontrar una tendencia clara hacia los sistemas con topología de malla en las propuestas analizadas. Esta inclinación se debe a que la topología de malla ofrece una mayor confiabilidad, ya que los pares, al tener varios receptores y proveedores, pueden tolerar con mayor facilidad la ausencia repentina de nodos o errores en las conexiones.

Basándose en lo anterior, la topología seleccionada para Pixtream es Malla.

3.8.2. Red P2P subyacente

De todas las propuestas analizadas, sólo dos se basan en una arquitectura P2P establecida: Bitorrent y GNUStream. En ambos casos las modificaciones necesarias para que estos sistemas puedan soportar *media streaming* de forma eficiente son extenuantes y requieren cambios importantes a los protocolos, tanto BitTorrent [9] como Gnutella [27]. Por esta razón se consideró que trabajar sobre una red P2P ya existente no provee ninguna ventaja estratégica en el desarrollo del proyecto y por lo tanto se decidió que Pixtream usará un protocolo y un sistema P2P propio.

3.8.3. Ingreso a la red o *bootstrapping*

El mecanismo usado en el proceso de ingreso a la red es uno de los más heterogéneos dentro de las plataformas estudiadas. Mientras que para algunos trabajos como BitTorrent para *streaming* y BEAM se utiliza un rastreador independiente de la fuente multimedia para este proceso, En otros sistemas, como AnySee y DONet, el proceso de *bootstrapping* es realizado por el mismo par que actúa como fuente de *streaming*. GNUStream se basa en una red P2P pura, completamente independiente, y por lo tanto el proceso de ingreso a la red es

completamente diferente. PULSE es flexible en el mecanismo a utilizar, permitiendo que se use un sistema de rastreador o un algoritmo tipo GOSSIP.

Con el objetivo de no sobrecargar la labor del par fuente y debido a que la implementación de un sistema P2P puro completamente descentralizado es costosa en tiempo y no representa un aporte fundamental a los objetivos de este proyecto, se decidió el uso de un rastreador para controlar el ingreso a la red en Pixtream y para transmitir información básica sobre la utilidad de cada par.

3.8.4. Esquema de distribución

Todas las propuestas analizadas, a excepción de AnySee, utilizan prácticamente el mismo sistema de división del flujo multimedia, el cual consiste en partir el flujo cada n bytes y generar un paquete con un número de secuencia y una estampa de tiempo. Adicionalmente, algunas propuestas como PULSE plantean la posibilidad de utilizar un paquete basado en un sistema de codificación de múltiples descripciones o MDC (*Multiple Description Coding* [42]).

Para Pixtream se seleccionó la división secuencial como modo de preparar el flujo multimedia para la distribución. Debido a las restricciones en el tiempo de desarrollo, alternativas como MDC no se aplicaron, sin embargo se proponen como trabajos futuros en el capítulo 8.3.

3.8.5. Sistema de búsqueda y publicación

Es necesario contar con un mecanismo que permita a los usuarios de la red localizar y acceder a contenido. La mayoría de las propuestas no plantean un sistema de búsqueda y acceso al contenido concreto. Muchos de ellos están diseñados para una sola sesión y, por lo tanto, no necesitan un sistema de búsqueda de contenido. Se decidió que Pixtream use un sistema similar, que permita múltiples sesiones completamente independientes utilizando diferentes instancias del rastreador y los pares por sesión. Así, para publicar una sesión de *streaming* se da la dirección del rastreador. Desde ahí se podrá acceder a la información relevante sobre lo que se va a transmitir. Los sistemas de búsqueda funcionaran de manera independiente, de la misma manera que se hace en BitTorrent.

3.8.6. Control de continuidad

Cuando se realiza *media streaming* en vivo, el contenido a compartir no está disponible desde el principio, sino que se va creando en tiempo real poco a poco. Teniendo en cuenta esto, las propuestas analizadas plantean, en su mayoría, un sistema de ventana deslizante, en el cual se utilice un rango de n partes del flujo multimedia que formen conjunto. Este conjunto de pedazos, serán aquellos que serán enviados y solicitados a otros pares por un cierto periodo de tiempo. A medida que el tiempo corre, esta ventana se va deslizando poco a poco, desde los pedazos más antiguos hasta los más nuevos, que han sido generados recientemente.

Debido a que el sistema de ventana deslizante es prácticamente un común denominador en las propuestas estudiadas, se decidió usarlo para Pixtream.

3.8.7. Componentes

El rango de los componentes o aplicaciones separadas que se usan en las propuestas de sistemas de *media streaming* P2P varía bastante. Esto depende, en gran medida, de la red P2P subyacente que se haya escogido para un sistema en particular. En el punto 3.8.3, se decidió que se iba a usar un rastreador para el proceso de ingreso a la red en Pixtream. El rastreador es, entonces, el primer componente o aplicación de una red Pixtream.

Para el resto de componentes del sistema, los planteamientos son más o menos uniformes entre las diferentes propuestas. Está claro que es necesario tener un par fuente que se conecta directamente con el flujo multimedia y que se encarga de dividirlo y empaquetarlo para ser enviado a través de la red P2P. Existe además un par normal, el cual no tiene acceso al flujo multimedia original, sino que, en cambio, recibe paquetes secuenciados en desorden y que debe encargarse de ordenarlos, unirlos y recomponer el flujo multimedia para que pueda ser procesado por un reproductor.

Pixtream, entonces, tiene tres componentes fundamentales:

- Rastreador
- Par fuente
- Par ordinario

3.8.8. Incentivos a la contribución

En una red P2P de este tipo, es crucial lograr que la mayoría de los pares contribuyan con sus recursos. Es de especial importancia la contribución en ancho de banda de salida para que se pueda explotar al máximo el potencial del P2P *streaming*.

Algunas propuestas como BitTorrent para *streaming*, utilizan el clásico algoritmo *tit-for-tat*, en el cual los nodos sólo envían paquetes a otros nodos de los cuales hayan recibido paquetes. En BEAM y PULSE se usan mecanismos para determinar qué tan útil está siendo un par en la red y proporcionar paquetes a aquellos con una mejor utilidad.

En todos los casos se realizan envíos aleatorios de paquetes a cierto porcentaje de solicitantes para evitar cuellos de botella al inicio de la transferencia. Otros sistemas como DONet, AnySee o GNUStreaml, no plantean un esquema claro de incentivo a la contribución de los pares.

Por lo anterior, Pixtream utilizará un sistema de cálculo del factor de utilidad de cada par, teniendo en cuenta el número de paquetes que otros pares recibieron de dicho par en un espacio de tiempo. Adicionalmente, la selección del destinatario de un determinado número de paquetes estará dada aleatoriamente, con el fin de evitar problemas iniciales de envío.

3.8.9. Control de la estructura de la red

En esta categoría se examinaron las propuestas con el objetivo de determinar qué entidad es la encargada de mantener la estructura de la red. Es decir, siendo que un sistema forme una red con una topología dada, quién o qué se encarga de mantener esa forma. Similarmente a cómo se observa en el caso de las topologías empleadas, se puede ver un común denominador en el control de la forma de la red en casi todas las propuestas estudiadas. En este caso común, se usa una estructura aleatoria o semi-aleatoria qué es determinada por decisiones individuales tomadas por cada par. La única excepción es nuevamente el caso de AnySee en dónde se usa el par fuente como controlador de la estructura de la red.

Las estructuras aleatorias y semi-aleatorias ofrecen un menor costo operativo ya que el control depende de las entidades individuales y, por consiguiente, la complejidad de manejar la red no crece linealmente junto con el número de nodos [11]. Adicionalmente, el número de mensajes de control que se tienen que enviar es mínimo. Por el contrario, un sistema controlado por una entidad central debe realizar muchos más cálculos y enviar más mensajes a medida que la red crece, como es el caso de los sistemas basados en topologías de múltiple árbol.

Teniendo en cuenta lo anterior, el control de la estructura de la red en Pixtream estará dado principalmente por los pares individuales, quienes serán los encargados de decidir con qué otros pares intercambiar información. De manera similar a como funcionan sistemas como BEAM o PULSE, existirán adicionalmente "sugerencias" qué podrán ser dadas a los nodos sobre a qué pares escoger, pero serán estos últimos los que decidirán qué hacer en última instancia. Esto con el objetivo, por ejemplo, de lograr que los nodos que mejor contribuyen estén más cerca de la fuente de datos.

3.8.10. Política de recepción de pedazos

La política de recepción de pedazos se refiere a qué algoritmo usa el sistema para determinar a qué pares debe solicitar pedazos del flujo multimedia. El rango de estrategias utilizadas en las propuestas analizadas es grande. El punto en común lo tienen BitTorrent y DONet que utilizan el algoritmo *rarest-first*. El cual consiste en solicitar aquellos pares que sean más raros dentro de la red, es decir, que sean poseídos por menos pares. Esta estrategia es muy importante para evitar que muchos pares se "peleen" por un pedazo que sólo sea poseído por uno o pocos pares.

Otras propuestas como PULSE y GNUstream, no utilizan un algoritmo concreto para solicitar pedazos a otros pares, sino que lo solicitan de cualquiera que lo tenga disponible.

Debido a la importancia que puede tener para la red el hecho de que los pedazos estén distribuidos de la forma más heterogénea posible, y con esto evitar cuellos de botella en los cuales muchos pares soliciten unos pedazos poseídos por unos pocos, se decidió que Pixtream debe utilizar el algoritmo *rarest-first*.

3.8.11. Política de envío de pedazos

Cuando un par realiza una petición a otro par, solicitando un pedazo del flujo multimedia, el otro par debe decidir si envía o no el paquete. Para tomar dicha decisión, el par debe utilizar un algoritmo de envío de paquetes.

En las propuestas analizadas, el uso de algoritmos para determinar el envío de pedazos varía completamente de sistema en sistema. Cada estrategia tiene pros y contras. Se determinó que para lograr un mejor rendimiento, es importante aplicar varias estrategias y no sólo una. Es importante por ejemplo utilizar un envío basado en el factor de utilidad de cada par, para incentivar la contribución como se mencionó en el punto 3.8.8. Además, es importante enviar los pedazos de acuerdo con un indicativo de prioridad con respecto al tiempo de distribución, como se realiza en PULSE. Pixtream está basado en estas dos estrategias para tratar de garantizar la recepción de la mayor parte de los paquetes en una sesión de *streaming* multimedia.

3.8.12. Estrategia de asociación entre pares

De las propuestas analizadas, sólo una utiliza un esquema de asociación entre pares: BEAM. Para esta propuesta las alianzas son muy importantes ya que determinan la política de petición y envío de pedazos. Debido a que los algoritmos que se usan en Pixtream para estos dos aspectos son diferentes, no hay un buen motivo para usar un sistema de asociación entre pares en Pixtream.

Capítulo 4

Arquitectura

4.1. Resumen de las características seleccionadas

En el capítulo 3 se realizó una comparación de las distintas propuestas de sistemas de *media streaming* P2P. De esa comparación se seleccionaron las características que deberían ser implementadas en Pixtream. La tabla 4.1 resume dichas características.

Topología:	Malla.
Red subyacente:	Ninguna existente. Una propia.
Ingreso a la red:	Rastreador.
Esquema de distribución:	División secuencial del flujo multimedia.
Búsqueda y publicación:	Múltiples instancias del rastreador.
Control de continuidad:	Ventana deslizante.
Componentes:	Rastreador, par fuente y par normal.
Incentivos:	Cálculo del factor de utilidad de cada par.
Control de la estructura:	Independiente por cada par, con sugerencias para posicionar a los pares con mayor factor de utilidad.
Política de recepción:	Algoritmo *rarest-first*.
Política de envío:	Envío basado en factor de utilidad de los pares y prioridad de los paquetes solicitados.
Asociaciones entre pares:	Ninguna.

Tabla 4.1: Resumen de las características seleccionadas en Pixtream

4.2. Visión general del sistema

En un sistema de *media streaming* tradicional, basado en una arquitectura cliente/servidor, existen dos componentes importantes: El servidor de *streaming* y el reproductor multimedia. Estos componentes suelen ser bastante complejos. Por lo general tienen que realizar todas las operaciones requeridas para el manejo multimedia como codificación, decodificación, compresión, descompresión y sincronización. El objetivo de Pixtream no es lidiar con todas estas operaciones propias de los contenidos multimedia, sino que se enfoca exclusivamente al sistema de transporte que se usa para hacer que el contenido multimedia llegue desde su fuente hasta el espectador.

Para lograr que Pixtream opere sólo en el proceso de transportar contenido, olvidándose de las operaciones multimedia complejas, se decidió que Pixtream debería actuar en la mitad de un sistema tradicional de *streaming* cliente/servidor. Para esto, Pixtream deberá tener componentes que actúen como un cliente multimedia y como un servidor multimedia al mismo tiempo.

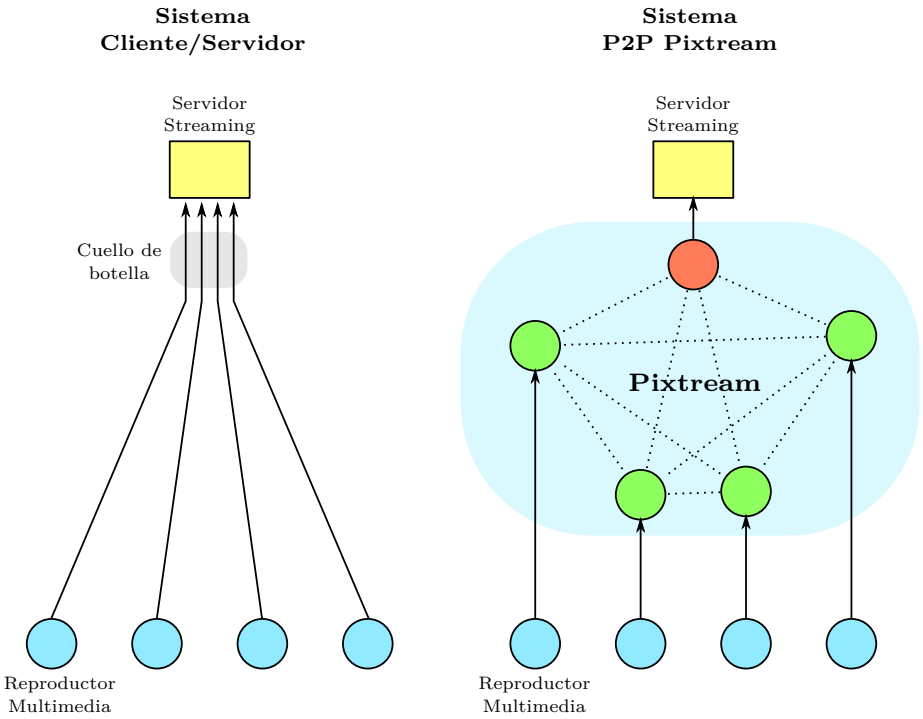


Figura 4.1: *Streaming* tradicional frente a *streaming* P2P.

La figura 4.1 contrasta la forma en la que funcionan un sistema de *media streaming* basado en arquitectura cliente/servidor tradicional y Pixtream. Como se puede observar, Pixtream actúa en el medio del reproductor multimedia y el

servidor. Gracias a esto, se evita la necesidad de desarrollar y reinventar estos dos componentes. Además, el número de conexiones salientes desde el servidor se reduce a una, logrando el objetivo básico de utilizar redes P2P para *media streaming*: reducir el cuello de botella en el uso de ancho de banda en esta parte del sistema.

Como se mencionó en la selección realizada en la tabla 4.1 (página 30), Pixtream contará con dos tipos de pares diferentes: El *par fuente* y el *par normal*. Al analizar Pixtream como una entidad que existe en medio de un servidor de *streaming* y un reproductor multimedia, se pueden entender mejor los roles de estos dos tipos de pares: El *par fuente* es aquel que se conecta con el servidor de *streaming*, mientras que el *par normal* es aquel a quien se debe conectar el reproductor multimedia para acceder al contenido.

El *par fuente* es el único de la red que tiene acceso al flujo multimedia original proveniente del servidor de *streaming*. En una red Pixtream debe existir un sólo *par fuente*. Este también tiene la tarea de dividir y secuenciar el flujo multimedia en pedazos que van a ser distribuidos en diferente orden por toda la red P2P.

Todos los demás pares en una red Pixtream son pares normales. Estos se dedican a solicitar y enviar pedazos a otros pares. Adicionalmente, tienen el trabajo de unir los pedazos consecutivos y recrear el flujo multimedia tal cómo lo recibió el *par fuente*. Posteriormente el par debe actuar como un servidor de *streaming* en sí mismo con el objetivo de que un reproductor multimedia pueda conectarse a él y acceder al contenido tal y como si viniera del servidor original.

Adicionalmente al *par fuente* y al *par normal*, existe otro componente muy importante en una red Pixtream: el *rastreador*. El objetivo principal del *rastreador* es el de actuar como puerta de entrada para todos los demás pares. Antes de que un par pueda participar en una red Pixtream, debe primero contactar al *rastreador*, el cual le entregará una lista de los otros pares que están en la red. Posteriormente, el par podrá empezar a entablar conexiones con otros y compartir información. Además de esto, el *rastreador* también sirve para recopilar estadísticas del rendimiento de cada par y ofrecer dicha información a otros pares con el objetivo de facilitar sus decisiones respecto a con quién compartir pedazos del flujo multimedia.

La figura 4.2, muestra una descripción general del funcionamiento de una red Pixtream. En la figura se puede observar las conexiones que tiene que generar cada componente para crear los enlaces necesarios para el funcionamiento de la red. El *par fuente* tiene que generar un enlace con el servidor de *streaming* para acceder al contenido multimedia. Así mismo, cada par tiene que esperar un enlace proveniente de un reproductor multimedia para poder visualizar y/o escuchar el contenido. Adicionalmente, cada par debe establecer enlaces P2P con otros pares. Y para lograr este objetivo, cada par tiene que contactar un *rastreador* que le informa sobre la presencia de los demás.

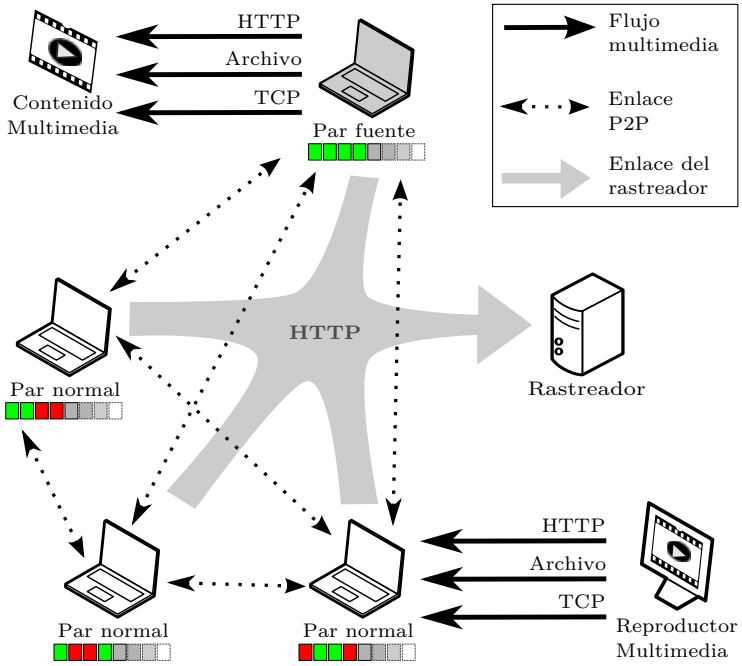


Figura 4.2: Enlaces en un sistema Pixtream.

4.3. Requisitos

Los requisitos para implementar un prototipo *software* de un sistema de *media streaming* P2P se dividen en dos categorías de acuerdo con las recomendaciones de Bass *et al* [43]: Requisitos funcionales y no funcionales.

4.3.1. Requisitos funcionales

1. Crear un sistema de comunicación P2P que contenga un sistema de entrada y un protocolo de comunicación entre pares.
2. Capacidad para conectarse a un servidor de *streaming* o algún tipo de contenido multimedia.
3. Capacidad para actuar como un servidor de *streaming* para que un reproductor pueda acceder al contenido.
4. Dividir el contenido en múltiples pedazos.
5. Unir múltiples pedazos del contenido multimedia para formar el flujo original.
6. Gestionar envíos y peticiones de paquetes entre pares.

7. Interfaz de usuario gráfica que permita un uso más sencillo de la aplicación a todo usuario

8. Interfaz de usuario de línea de comandos para usuarios avanzados y otros programas.

9. Obtener estadísticas e información sobre la red.

4.3.2. Requisitos no funcionales

1. El sistema debe poder escalar, debe poder adaptarse a unos pocos pares o a muchos sin recargar el servidor de *streaming*, del cual se obtienen los datos.

2. Flexibilidad en el sistema de conexión al contenido de *streaming*. El sistema debe permitir fácilmente añadir nuevos esquemas y protocolos de comunicación con servidores de *streaming* y otros tipos de contenidos multimedia.

3. Soporte multiplataforma. El programa deberá funcionar en varios sistemas operativos.

4. Código y distribución libre. El programa tendrá una licencia libre [1] para ser usado y mejorado por cualquier persona.

4.4. Diseño arquitectónico

Como se discutió en secciones anteriores, un sistema Pixtream se basa en tres componentes fundamentales: El *rastreador*, el *par fuente* y el *par normal*. Cada componente es una aplicación separada y cada uno tiene una arquitectura.

La figura 4.3 describe las interacciones de cada componente. como se puede ver el *par fuente* obtiene el flujo multimedia de un servidor de *streaming* e interactúa con los pares normales a través de un enlace P2P. El *par normal* también interactúa con otros pares normales y sirve como servidor al reproductor multimedia. Ambos tipos de pares interactúan con el *rastreador* que sirve de enlace para toda la red P2P.

A continuación se describe cada componente de Pixtream en detalle y se explica el funcionamiento de cada una de sus partes y las interacciones entre ellas.

4.4.1. Arquitectura del *rastreador*

El *rastreador* de Pixtream es el componente que sirve de puerta de entrada para los pares en la red. El objetivo principal del *rastreador* es mantener una lista de los pares que están participando de una sesión de *streaming* con Pixtream y darlos a conocer a todo par nuevo que desee ingresar a la red.

[1] Libre en el sentido de la *Free Software Fundation* [44]

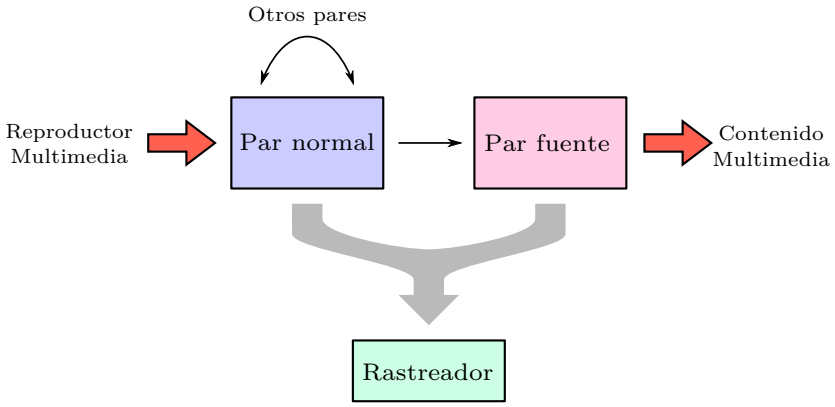


Figura 4.3: Componentes de Pixtream

El *rastreador* se comunica con los pares a través del protocolo HTTP. En esencia es una aplicación web, aunque no necesariamente para ser accedida a través de un navegador, sino principalmente para ser consultada por los pares. La arquitectura del *rastreador* es simple y está basada en tres capas: Capa de control, capa de presentación y capa de conexión.

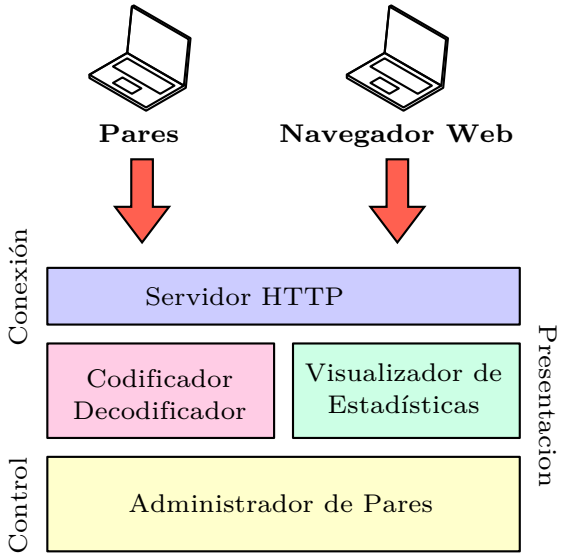


Figura 4.4: Arquitectura del rastreador de Pixtream

El *rastreador* puede ser accedido de dos formas. Principalmente es accedido por pares que estén en la red Pixtream o deseen ser parte de ella. Adicionalmen-

te, el *rastreador* también puede ser consultado por un navegador web u otras aplicaciones web para que un usuario pueda conocer el estado de una sesión de *streaming*. De esta manera, sería posible para un usuario conocer qué contenido se está transmitiendo en la sesión y cuantos usuarios están conectados al sistema. Esto se constituye en la forma básica de publicación de contenido en Pixtream y la base para que un usuario decida hacer o no parte de una sesión. También es una valiosa herramienta para recolectar información estadística sobre el estado de una sesión de *streaming* que sirva para un posterior análisis.

Administrador de pares

La capa de control administra los pares de una sesión Pixtream. Los objetivos del administrador de pares son:

1. Mantener una lista de todos los pares en una red Pixtream.
2. Llevar registro de la actividad de los pares, eliminando de la lista aquellos que se desconecten o se vuelvan inactivos.
3. Calcular el factor de utilidad de un par basándose en las contribuciones en recursos que este haya hecho a la red.
4. Informar a los pares en la red, especialmente al *par fuente*, del desempeño de otros pares para que estos lo tengan en cuenta al momento de determinar a qué pares enviar paquetes.
5. Mantener información de la sesión de *streaming* y enviarla a los pares y a los usuarios del sistema.

Módulo de visualización

El módulo de visualización de estadísticas genera una presentación en HTML lista para ser visualizada por un navegador web tradicional. Esto permite a los usuarios posibles clientes de una sesión de streaming obtener información acerca de la sesión antes de unirse a ella.

Módulo de codificación/decodificación

El módulo codificador/decodificador, de la misma manera que el módulo de visualización, sirve para presentar información de una sesión de *streaming*, sin embargo, en vez de presentar la información para mostrarse en un navegador, se presenta para ser entendida por los clientes. Por lo tanto, se utiliza un formato de presentación más compacto debido que tiene que ser accedido con mayor frecuencia. No es necesario que el contenido sea fácilmente legible por personas, ya que es la aplicación cliente la que va a acceder a esta información y no los usuarios.

Módulo de conexión

El módulo de conexión contiene un servidor HTTP incorporado que procesa las peticiones hechas por los pares y los navegadores web y presenta la información requerida usando los módulos anteriormente descritos.

4.4.2. Arquitectura del *par normal*

El *par normal* es el componente principal de Pixtream. Lo normal en una sesión de *streaming* es que existan varias instancias de este componente, todas interactuando entre sí. La arquitectura de un *par normal* está formada por tres capas: Conexión, Control e Interfaz. Cada capa tiene a su vez varios módulos y submódulos.

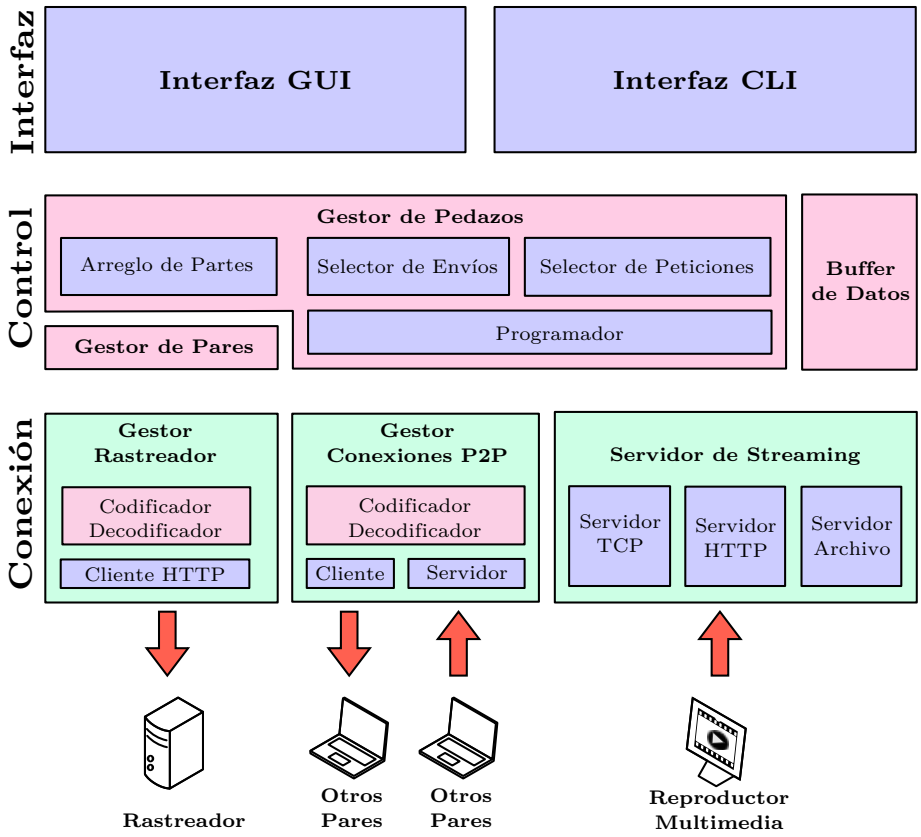


Figura 4.5: Arquitectura del *par normal* de Pixtream

Capa de conexión

La capa de conexión se encarga de controlar la interacción del par con otras entidades a través de la red. Pixtream es un sistema distribuido complejo y, por lo tanto, existen varias formas de comunicación por red entre procesos y programas independientes. Esta capa tiene tres módulos: El gestor de *rastreador*, el gestor de conexiones P2P y el servidor de *streaming*.

La figura 4.5, describe la arquitectura de un *par normal* en Pixtream. Las flechas rojas indican las conexiones entrantes o salientes que establece el par y las entidades con las cuales se establecen las conexiones.

Gestor de *rastreador*

El gestor de *rastreador*, como su nombre lo indica, es el módulo encargado contactar al par con el *rastreador* del sistema. En la base de la conexión, este módulo utiliza un cliente HTTP para establecer conexión con el *rastreador* y obtener una lista pares que se encuentran en la red. Esta lista está codificada por el *rastreador*, por lo cual el módulo tienen un submódulo de codificación y decodificación que usa para posteriormente poder pasar la información a la capa de control. Adicionalmente, el módulo también se encarga de anunciar periódicamente la actividad al *rastreador* para que este no tome al par como inactivo. A través del *rastreador* también se recibe información sobre el rendimiento de otros pares en la red.

Gestor de conexiones

El gestor de conexiones P2P se encarga de establecer y recibir conexiones desde y hacia otros pares en la red. Una aplicación P2P es, por definición, una aplicación de red que actúa al mismo tiempo como servidor y como cliente [45]. Consecuentemente, Pixtream debe actuar de las dos formas también. Por esta razón, el módulo de conexiones P2P tiene en su base dos partes: el cliente y el servidor. El ciente se dedica a establecer conexiones con otros pares y el servidor abre un puerto para esperar a que otros pares entablen conexiones con si mismo. La información transmitida por las conexiones entrantes y salientes está codificada de acuerdo con el protocolo Pixtream (véase capítulo 5). El manejo de las conexiones tanto salientes como entrantes es casi idéntico salvo por la forma en la que se entabla la conexión.

Servidor de *streaming*

El servidor de *streaming* es el módulo que se encarga de permitir las conexiones entrantes de un reproductor multimedia. Este módulo es el que permite que se pueda reproducir el contenido transmitido durante la sesión de *streaming*. El objetivo básico de este módulo es imitar el comportamiento de un servidor de *streaming* tradicional, al que usualmente se conecta un reproductor multimedia. Debido a que existen varios protocolos de *streaming* soportados por distintos servidores y reproductores, el módulo cuenta con varios submódulos para cada

protocolo. Un objetivo importante del módulo es proveer de una interfaz común para todos los submódulos de protocolos, permitiendo que puedan ser agregados nuevos submódulos para el soporte de otros protocolos. En la primera versión de Pixtream se conciben tres protocolos de *streaming* soportados: HTTP, TCP crudo y un servidor figurado que en realidad salva el contenido a un archivo del sistema operativo. En la sección 8.3 (página 95) se nombran algunos otros protocolos que pueden ser usados en trabajos futuros.

Capa de control

La capa de control es el cerebro de un par en Pixtream. En esta capa se realizan todas las operaciones principales, se toman las decisiones pertinentes y se mandan las órdenes a la capa de conexiones. También se presenta la información del par para que la capa de interfaz pueda presentarlas al usuario y controlar ciertas opciones.

Las funciones de la capa de control son:

1. Controlar información acerca de la sesión de *streaming*. Qué contenido se está transmitiendo y cuál es el estado de la transmisión.
2. Controlar el proceso de transferencia. Pausar y continuar.
3. Llevar una lista de los pares compañeros que están disponibles, el estado y rendimiento de los mismos.
4. Mantener un control de los pedazos que se tienen, los que se necesitan y controlar el sistema de ventana deslizante y de prioridades.
5. Seleccionar los pares compañeros a los cuales enviar peticiones para que provean de pedazos que se necesiten.
6. Seleccionar los pares compañeros a los cuales se les va a enviar paquetes de acuerdo con sus peticiones.
7. Programar el envío periódico de mensajes a otros pares.
8. Unir los pedazos consecutivos conseguidos para reconstruir el flujo multimedia original y así alimentar al servidor de *streaming* en la capa de conexión.
9. Proveer información del estado del par y los eventos que ocurren para que pueda ser accedida por la capa de interfaz.

La capa de control tiene tres módulos: El gestor de pares, el gestor de pedazos y el *buffer* de datos.

Gestor de pares

El gestor de pares almacena y administra información relacionada con los otros pares de la red. Esta información proviene de la capa de conexiones, en el gestor de *rastreador*. Dentro del gestor de pares, estos pueden estar en varias categorías: Aquellos de quienes se tiene conocimiento, pero no se ha entablado conexión todavía con ellos. Aquellos con quien se ha entablado conexión pero están bloqueados y aquellos con quien se tiene conexión y no están bloqueados. También se almacena información sobre el factor de utilidad de cada par, para luego tenerlo en cuenta a la hora de mandar peticiones o envíos.

Gestor de pedazos

El gestor de pedazos es el módulo más grande de esta capa. Como su nombre lo indica, se encarga de administrar los pedazos del flujo multimedia. El módulo está compuesto por cuatro submódulos: Arreglo de partes, selector de envíos, selector de peticiones y el programador de comunicación.

Arreglo de partes

El arreglo de partes mantiene una lista de los pedazos del flujo que se han descargado y los que faltan por descargar. El módulo también guarda y administra información referente a qué pares compañeros tienen qué pedazos disponibles, de qué pares se han descargado los pedazos ya obtenidos. El módulo también se encarga de controlar el sistema de ventana deslizante.

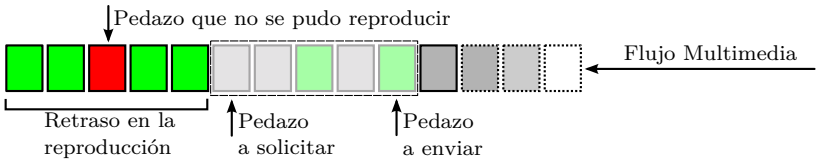


Figura 4.6: Ventana deslizante en Pixtream

La figura 4.6 ilustra el funcionamiento de la ventana deslizante en Pixtream. Debido a que el flujo multimedia en un sistema de *streaming* es algo que se va generando a medida que pasa el tiempo, el intercambio y división de pedazos no puede funcionar en todo el flujo, porque sino habría que generarlo completamente antes de transmitirlo y se pierde el sentido de transmisión en tiempo real del *media streaming*. Por esta razón se debe escoger una sección del flujo multimedia, con un número n de pedazos que represente las partes que se van a intercambiar. A medida que avanza la transmisión, los pedazos más antiguos se van dejando atrás, fuera de la ventana deslizante y otros nuevos que se van generando entran a esta. De esta forma, se puede decir que la ventana se va "corriendo" poco a poco desde los pedazos más antiguos, hasta los más nuevos. Si un pedazo dentro de la ventana deslizante no ha podido ser conseguido al

momento de que esta se corra hacia delante, entonces ese pedazo debe ser descartado debido a que la reproducción multimedia debe continuar y es preferible tener un pequeño salto en la reproducción que un corte mientras se consiguen los pedazos requeridos.

Módulo de programación

El submódulo de programación de la comunicación (Programador)se encarga de mantener una cola de mensajes que deben ser enviados a otros pares. Este componente es el que se encarga de ordenar a la capa de conexiones qué mensajes debe enviar a otros pares. En este módulo se mandan los mensajes básicos que anuncian la supervivencia de un par a otros y también se programan los envíos y peticiones que se hacen a otros paquetes. Adicionalmente, se controlan los estados de bloqueo y desbloqueo hacia otros pares y la tasa a la cual se envían los paquetes. Esto se hace con el objetivo de controlar y llevar un registro del uso de ancho de banda. El programador anota detalles de qué pares han solicitado paquetes y a quienes se han enviado.

Selector de envíos

El selector de envíos evalúa la lista de peticiones que han hecho otros pares y se encarga de decidir a quiénes se les envía pedazos o a quienes se rechaza. Este componente utiliza las políticas que se seleccionaron basándose en el estudio de las propuestas de *media streaming*, mencionadas en la sección 3.8.

Selector de peticiones

El selector de peticiones analiza los paquetes que son necesarios para la reproducción y decide, de entre los pares que los tenga, a cuál sería más conveniente enviar una petición. El selector también debe tener en cuenta solicitudes fallidas en las que un par no responda y buscar un plan de contingencia. Al igual que el selector de envíos, este componente se basa en las políticas seleccionadas en la sección 3.8.

Buffer de datos

El *buffer* de datos toma los pedazos obtenidos que se encuentran en el arreglo de partes, los ordena y los une para formar un flujo secuencial tal como lo recibió el *par fuente*. Este flujo es luego pasado a la capa de conexión para que sea servido mediante el servidor de *streaming*.

Capa de interfaz

La capa de interfaz se encarga de presentar el contenido y funcionamiento de un par Pixtream al usuario. Esta capa tiene dos módulos: Interfaz GUI (Graphical User Interface) e Interfaz CLI (Command Line Interface).

Interfaz gráfica

La interfaz gráfica está diseñada para facilitar el uso del programa a los usuarios comunes. De esta manera, se encontrarán con elementos comunes en una interfaz de usuario como botones, ventanas, etc.

Interfaz de línea comandos

La interfaz de línea de comandos está diseñada para usuarios avanzados que quieran usar el programa desde una terminal de comandos. Esta interfaz también permite al programa del *par fuente* ser utilizado por otros programas, por ejemplo *scripts* o *plugins* de un reproductor multimedia o programas para automatizar las pruebas del sistema (ver 7).

4.4.3. Arquitectura del *par fuente*

El *par fuente* funciona como una extensión del *par normal*. Un *par fuente* contiene todos los elementos que un *par normal*, pero con un par de módulos extra.

La figura 4.7 describe la arquitectura de un *par fuente*. Como se puede observar, varios componentes son los mismos que ya se mencionaron en la arquitectura del *par normal*. En la zona delineada con amarillo, se pueden ver los dos componentes adicionales que agrega el *par fuente*: El divisor de flujo multimedia y el cliente de *streaming*.

Cliente de streaming

El cliente de *streaming*, funciona en la capa de conexión y es el componente encargado de recibir el contenido multimedia original. Este componente debe tener la capacidad de conectarse con un servidor de *streaming* tradicional o cualquier otra fuente multimedia. El módulo a su vez está compuesto por varios submódulos, cada uno de los cuales está diseñado para conectarse al servidor de *streaming* usando diferentes protocolos.

Divisor del flujo multimedia

El divisor de flujo multimedia es el componente que toma el contenido recibido por el cliente de *streaming* y lo divide en varios pedazos. Cada pedazo recibe una secuencia y una estampa de tiempo que lo identifica. En el par fuente, el arreglo de partes es alimentado directamente por el divisor de flujo, ya que el *par fuente* no recibe paquetes de otros pares.

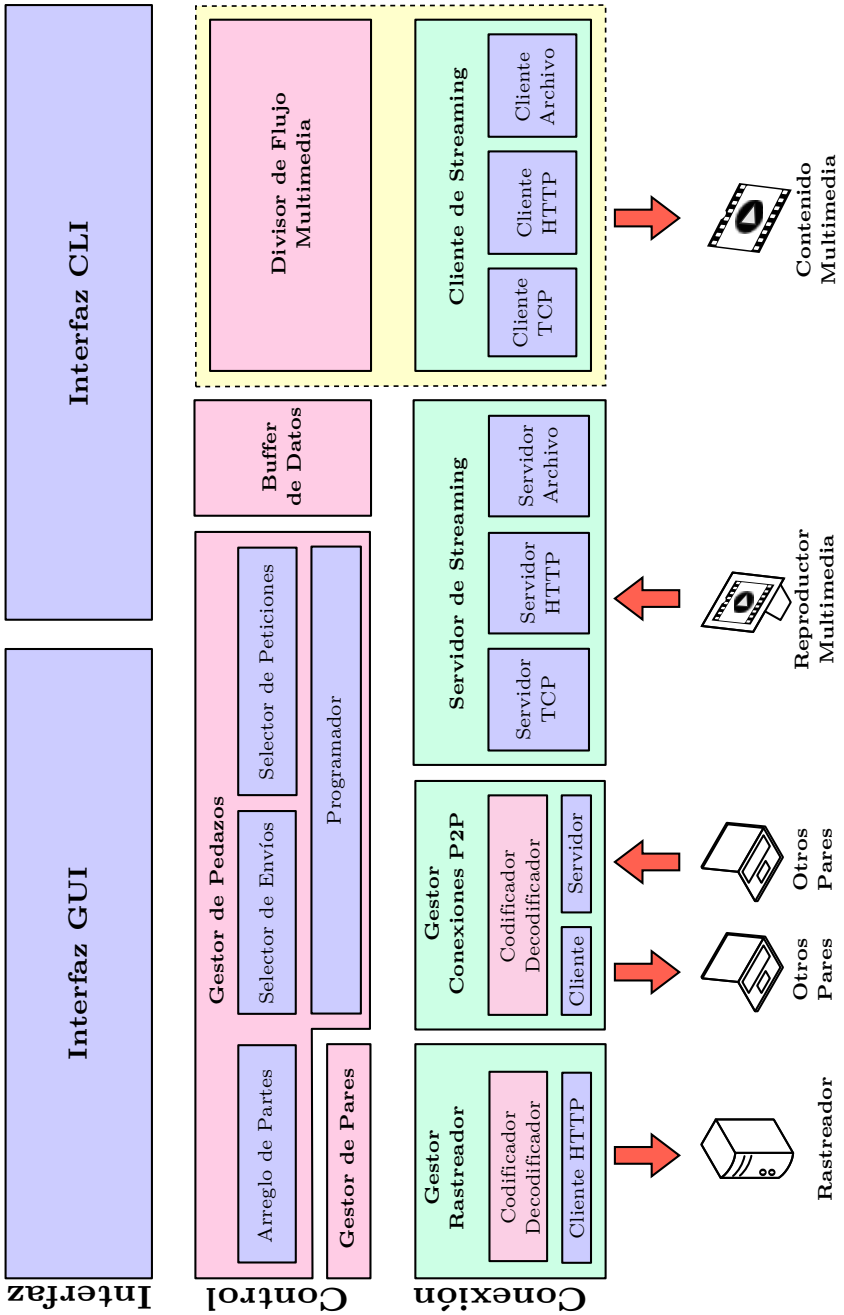


Figura 4.7: Arquitectura del *par fuente* de Pixtream

Capítulo 5

Especificación del protocolo

El protocolo Pixtream contiene todas las reglas y métodos de comunicación usados en un sistema P2P Pixtream. Usando esta especificación de protocolo es posible crear otras implementaciones de Pixtream que sean compatibles.

Esta especificación se divide en dos partes: El protocolo del rastreador y el protocolo de la red de pares. El protocolo describe la comunicación con el rastreador, tanto de los pares como de otros servicios. El protocolo de la red de pares describe la forma en que se comunican los pares unos con otros.

5.1. Protocolo de *rastreador*

Un *rastreador* de Pixtream es un servicio que funciona sobre el protocolo HTTP[1] y responde a peticiones GET. Un *rastreador* puede funcionar como una aplicación web. El *rastreador* es accedido por los pares a través del protocolo HTTP pero también puede ser accedido a través de un navegador web o otros sitios web para obtener estadísticas de una sesión.

Un par se comunica con el *rastreador* enviando una serie de datos de entrada y recibiendo datos de salida producidos como un documento HTTP.

5.1.1. Datos de entrada

El *rastreador* obtiene los datos de entrada a través de peticiones HTTP GET. Las peticiones se realizan a través de varias URLs provistas por el *rastreador*, las cuales vienen acompañadas de varios parámetros GET dependiendo de la URL y la función que se solicite.

Los parámetros de la petición son agregados a la URL usando una cadena de consulta o *query string* como se define en el estándar RFC-3986 [47]: agregando un signo de interrogación (`?`) al final de la base de la URL y agregando cada parámetro de la forma `parametro=valor` separados por un *ampersand* (`&`).

[1] *Hypertext Markup Language* es el protocolo de la web definido por el estándar RFC-2616 [46]

Los parámetros de entrada del *rastreador* deben estar adecuadamente escapados, es decir, que deben usarse sólo los caracteres permitidos en una URL o de otra forma usar la convención de escape `%nn`. Para más información acerca de la forma en que debe ser definida la URL se debe revisar el estándar RFC 1738. [48].

5.1.2. Datos de salida

Para la codificación de los datos de salida se usa el formato JSON[2]. La razón por la cual se decidió su uso es que este es un formato bastante extendido y estandarizado, con soporte para muchos lenguajes de programación, adicionalmente, utiliza relativamente poco espacio para la representación de los objetos. Otros formatos como XML pueden servir para el mismo propósito, sin embargo, se decidió no usarlos debido a que los datos ocupan proporcionalmente más espacio que en JSON.

El formato JSON se puede utilizar para representar objetos, mapas de *hash*, listas, y tipos de datos básicos. La especificación completa se puede encontrar en el estándar RFC 4627 [49]. El listado 1 contiene un ejemplo básico de información codificada en el formato JSON.

```
{
     "firstName": "John",
     "lastName": "Smith",
     "address": {
         "streetAddress": "21 2nd Street",
         "city": "New York",
     },
     "phoneNumbers": [
         { "type": "home", "number": "212 555-1234" },
     ],
     "newSubscription": false,
     "companyName": null
 }
```

Listado 1: Ejemplo de salida JSON

5.1.3. Mensajes del *rastreador*

El *rastreador* cuenta con tres mensajes:

1. Anuncio: el cual es usado por los pares de la red Pixtream para anunciarse y obtener una lista de todos los pares en la red.

2. Información de la sesión: usado para obtener datos sobre el material que se está transmitiendo y el estado actual de la red.

[2] *JavaScript Object Notation* es un formato ligero para el intercambio de datos basado en el lenguaje JavaScript. JSON está estandarizado en RFC-4627 [49]

3. Reporte de *factor de utilidad*: que permite a los pares reportar información de con quién se comparten pedazos del flujo multimedia. Además permite obtener datos de otros pares.

Anuncio de par

El anuncio es el punto de entrada a una red P2P Pixtream. Cuando un cliente desea conectarse y empezar una sesión de *streaming* lo primero que debe hacer es mandar un anuncio. El *rastreador* responderá con una lista de otros pares en la red y con la información sobre la sesión.

Parámetros de entrada El anuncio se hace a través de la URL del *rastreador* utilizando el sufijo `announce`. Estos son algunos ejemplos de URL para el anuncio.

- `http://www.mytracker.com/announce`
- `http://localhost:8080/announce`
- `http://mytracker.com:8000/announce`

La tabla 5.1 describe los parámetros que deben mandarse a una URL de anuncio del *rastreador*.

Parámetro	Descripción
`peer_id`	Cadena de caracteres de 20 *bytes* indicando la identificación del par que está realizando el anuncio.
`port`	El número de puerto sobre el cual el par está "escuchando". Otros pares intentarán crear conexiones conectándose a este puerto.
`ip` (opcional)	Dirección IP del par que se está conectando. Este parámetro es opcional debido a que la dirección IP puede ser usualmente determinada examinando de dónde provino la petición HTTP hecha al *rastreador*. Sin embargo, este parámetro es necesario en el caso en el que el par anunciándose está accediendo a la Web usando un *proxy* HTTP, en cuyo caso la dirección de quién realiza la petición sería la del *proxy* y no del par en cuestión. Otro posible escenario sería uno en el cual tanto el par como el *rastreador* se encuentren bajo la misma pasarela NAT[3], por lo tanto, la dirección de la petición sería una dirección local y no de Internet, lo cual limitaría la conexión con otros pares.

Continua en la siguiente página...

[3]NAT (*Network Address Translation*) es un mecanismo utilizado por los *routers* IP para intercambiar paquetes entre dos redes que se asignan mutuamente direcciones incompatibles.

...Continuación de la tabla

Parámetro	**Descripción**
`exit` (opcional)	Si este parámetro está presente, el *rastreador* tomará el anuncio como un anuncio de salida. Esto quiere decir que el par pretende desconectarse de la red. Es recomendable que todos los pares que quieran abandonar una sesión de *streaming* Pixtream envíen un anuncio de salida para que los demás puedan estar al tanto de su ausencia.

Tabla 5.1: Parámetros de entrada del anuncio Pixtream

Este es un ejemplo de una URL de entrada para un anuncio Pixtream:

```
http://tracker.com/announce?peer_id=PX00010000000000060003&port=1000
```

Datos de salida La respuesta a una petición de anuncio será un documento en texto plano (especificado `text/plain` en la cabecera HTTP) con un diccionario en formato JSON con las siguientes claves:

Clave	**Descripción**
`failure_reason`	Si esta clave está presente en el diccionario, entonces ninguna otra clave debería estar presente. Significa que se ha producido un error. El valor de la clave será una cadena con la descripción del error del *rastreador*.
`request_interval`	El valor de esta clave es un número entero indicando la frecuencia en segundos a la cual los pares deben enviar peticiones de anuncio al *rastreador*. Si el valor es, por ejemplo, 30, los pares deberán enviar anuncios cada 30 segundos. Si pasado un determinado tiempo el *rastreador* no recibe anuncios de un par, este será eliminado de la lista de pares del *rastreador*.
`content_title`	Contiene una cadena con el título del contenido que se está reproduciendo en la sesión de *streaming*.
`content_description`	Contiene una cadena con la descripción del contenido que se está reproduciendo. Esta cadena puede ser un poco más larga que `content_title` y puede ser omitida después de la primera respuesta de anuncio para evitar gasto de espacio.

Continua en la siguiente página...

Estandar RFC-2766

...Continuación de la tabla

Clave	**Descripción**
`content_length`	Contiene un número entero con la duración esperada del contenido que se está transmitiendo en segundos. Si es un contenido en vivo esta clave puede omitirse. También puede omitirse después de la primera respuesta de anuncio para ahorrar espacio.
`content_type`	Cadena describiendo el tipo de contenido que se está transmitiendo, los posibles valores son: `audio`, `video` y `audiovideo`. El valor de esta clave puede omitirse después de la primera respuesta de anuncio para ahorrar espacio.
`content_format`	Cadena describiendo el formato del contenido multimedia. Sirve para identificar los contenedores y los *codecs* en los que está codificado. Un ejemplo de valor de esta clave es `MPEG-2 + AAC (MP4)`. Esta clave es opcional y puede omitirse después de la primera respuesta de anuncio.
`peers`	El valor de esta clave es una lista de todos los pares conectados a la sesión. Cada par está representado por un diccionario. La tabla 5.3 especifica las claves de dicho diccionario.

Tabla 5.2: Descripción de la salida de un anuncio Pixtream

Clave	**Descripcion**
`peer_id`	Contiene una cadena de texto de 20 *bytes* con la identificación del par.
`ip`	El valor debe ser una cadena que contiene la direccion ip del par.
`port`	Contiene un número entero con el puerto en el cual el par está "escuchando" y recibirá conexiones entrantes.
`utility_factor`	Contiene un número entero indicando el factor de utilidad de un par. Esta información será usada por los pares para decidir a quienes enviar paquetes. Para más información sobre el *factor de utilidad* revisar la sección 5.1.3.

Tabla 5.3: Descripción del diccionario que describe un par

Algunos ejemplos de la salida de una petición de anuncio:

```
{"failure_reason": "Peer ID not given"}
```

Listado 2: Ejemplo de la salida de un anuncio fallido.

```
{
   "peers":[
      {
         "ip":"127.0.0.1",
         "utility_factor":0,
         "port":60010,
         "id":"PX000100000000060010"
      },
...
      {
         "ip":"127.0.0.1",
         "utility_factor":0,
         "port":60008,
         "id":"PX000100000000060008"
      }
   ],
   "content_title": "Naruto Shippuden Episodio 134",
   "content_description": "Naruto Shippuden",
   "content_type": "audiovideo",
   "content_length": 0,
   "content_format": "Theora + Vorbis (OGG)",
   "request_interval": 10
}
```

Listado 3: Ejemplo de la salida de un anuncio exitoso.

Reporte de *factor de utilidad*

Otra de las funciones del *rastreador* Pixtream es mantener un control de la utilidad de cada par. Para lograr esto, el *rastreador* compone el factor de utilidad de cada par con base en los paquetes que este haya enviado a otros pares. Sin embargo, el *rastreador* no recibe esta información del par en cuestión, sino de otros pares a los cuales este hay enviado paquetes.

Cada par en la red debe reportar periódicamente de qué pares ha recibido pedazos del flujo multimedia y cuantos *bytes* ha recibido. En cada reporte, el *rastreador* calcula la suma de todos los *bytes* que un determinado par ha enviado basándose en los reportes de otros. Esto evita que los pares intenten hacer trampa reportando más paquetes de los que en realidad han contribuido.

Parámetros de entrada El reporte de *factor de utilidad* se hace a través de la URL del *rastreador* utilizando el sufijo `utility`. Estos son algunos ejemplos de URLs para reportar el *factor de utilidad*:

- `http://www.mytracker.com/utility`
- `http://localhost:8080/utility`
- `http://mytracker.com:8000/utility`

Los parámetros de la petición corresponden a todos los pares de los cuales se ha recibido datos y la cantidad de datos en *bytes* recibidos. Los datos corresponden sólo a pedazos del flujo multimedia y no a otros mensajes intercambiados en la red de pares (véase mensaje *DataPacket* en la sección 5.2.5 página 58).

Cada parámetro de la petición corresponde a la identificación de un par y su valor es un número entero mayor o igual a cero indicando el número de *bytes* que se han recibido del par.

Un ejemplo de una petición para reportar el *factor de utilidad* sería:

```
http://tracker.com/utility?PX000100000000060003=1024&PX000100
000000060002=4096
```

Datos de salida Los datos de salida del reporte del *factor de utilidad* son dos posibles: éxito o error. Si el *factor de utilidad* ha sido correctamente reportado, el resultado será simplemente la cadena *"success"* codificada en formato JSON. Ejemplo:

```
1 "success"
```

Listado 4: Salida exitosa de un anuncio de *factor de utilidad*

Si el reporte fue incorrecto, incluyendo por ejemplo pares no existentes o utilizando un formato erróneo, la salida será un diccionario con la clave `failure_reason` de la misma forma en que se hace en la petición de anuncio. Un ejemplo sería:

```
1 {"failure_reason": "No data provided"}
```

Listado 5: Salida de error de un anuncio de *factor de utilidad*

Información de la sesión

La petición de información de la sesión del *rastreador* permite conocer el estado de una sesión de *streaming* Pixtream. Esta petición está diseñada para ser utilizada por otros servicios o para ser visualizada por un navegador web.

Entre los datos que se pueden utilizar se encuentran: número de pares conectados a la sesión, tiempo de la sesión, información sobre el contenido de la sesión entre otros.

Parámetros de entrada La información se puede obtener realizando una petición al *rastreador* utilizando la URL con el sufijo `info`. Por ejemplo:

- `http://www.mytracker.com/info`
- `http://localhost:8080/info`
- `http://mytracker.com:8000/info`

Datos de salida La salida de la petición de información es un diccionario codificado en JSON con las siguientes claves:

Clave	Descripción
`content_title`	Contiene una cadena con el título del contenido que se está reproduciendo en la sesión de *streaming*.
`content_description`	Contiene una cadena con la descripción del contenido que se está reproduciendo.
`content_length`	Contiene un número entero con la duración esperada del contenido que se está transmitiendo en segundos. Si es un contenido en vivo esta clave puede omitirse.
`content_type`	Cadena describiendo el tipo de contenido que se está transmitiendo, los posibles valores son: `audio`, `video`, `audiovideo`.
`content_format`	Cadena describiendo el formato del contenido multimedia. Sirve para identificar los contenedores y los *codecs* en los que está codificado. Un ejemplo de valor de esta clave es `MPEG-2 + AAC (MP4)`.
`peer_number`	Valor entero con el número de pares que actualmente están participando de la sesión.
`elapsed_time`	Valor entero con el número de segundos que han transcurrido desde que se inició la sesión de *streaming*.
`avg_transfer_rate`	Valor entero indicando el promedio de la tasa de transferencia en *bytes* por segundo.
`peer_ids`	Valor correspondiente a una lista de las identificaciones de cada par actualmente participando de la sesión.

Tabla 5.4: Parámetros de entrada del anuncio Pixtream

Un ejemplo de la salida de una petición de información es:

```
{
   "content_title": "Naruto Shippuden Episodio 134",
   "content_description": "Naruto Shippuden",
   "content_type": "audiovideo",
   "content_length": 0,
   "content_format": "Theora + Vorbis (OGG)",
   "elapsed_time": 625,
   "peer_number": 15,
   "avg_transfer_rate": 16423,
   "peer_ids": [
                "PX000100000000060001",
                "PX000100000000060002",
                ...
                "PX000100000000060015",
               ]
}
```

Listado 6: Ejemplo de la salida de una petición de información de pares

5.2. Protocolo de la red de pares

Esta sección describe el protocolo que usan los pares para comunicarse entre sí e intercambiar pedazos del flujo multimedia. La red de pares utiliza un protocolo diferente para comunicarse debido a que tiene prioridades diferentes: velocidad y flexibilidad.

La velocidad es un aspecto crítico de una red P2P. En Pixtream, un par tiene que entablar conexiones con decenas de otros pares en una sola sesión. Por lo tanto, la comunicación entre ellos tiene que ser lo más concisa posible; ya que de otra forma se estaría gastando ancho de banda que es necesario para transmitir el flujo multimedia. Es por esta razón que se debe usar un formato binario muy conciso para los mensajes.

La flexibilidad también es importante. Los pares deben ser capaces de comunicarse usando diferentes puertos debido a que pueden existir restricciones para ello.

5.2.1. Generalidades

El protocolo de comunicación entre pares se basa en TCP. Cada par de la red recibe una lista de otros pares del *rastreador* y mantiene un puerto TCP abierto para esperar conexiones. Un enlace entre dos pares ocurre cuando uno de los dos contacte al otro. Sea porque un par esté escuchando y reciba una conexión de otro o viceversa; lo que suceda primero.

Cada par es responsable de mantener una lista de las conexiones con otros pares, teniendo en cuenta la identificación de cada par, su dirección IP y el puerto sobre el que escucha. Los pares deben estar preparados para una desconexión intempestiva de cualquier par.

5.2.2. Estructura básica de los mensajes

Los mensajes que envían los pares para comunicarse entre sí están basados en un formato binario bastante simple. Cada mensaje consiste en una cadena de *bytes* de un cierto tamaño compuesta por varios campos seguidos. Cada campo tiene un tipo de datos. Pixtream tiene únicamente tres tipos de datos:

1. `unsigned int`: Corresponde a un entero mayor o igual a cero. Es representado por cuatro *bytes* utilizando la convención *big endian* [50]
2. `string`: Una cadena de *bytes* de tamaño fijo.
3. `bytes`: Una cadena de *bytes* de tamaño indefinido. Un campo de este tipo de datos siempre tiene que estar al final y sólo puede haber uno por mensaje.

Todos los mensajes tienen un prefijo compuesto de dos partes. Los primeros cuatro *bytes* de cada mensaje son un entero e indican el tamaño del mensaje. El quinto *byte* es un identificador del mensaje. Cada tipo de mensaje tiene una estructura diferente. En las secciones posteriores se detalla la estructura de cada mensaje.

Si un par recibe un mensaje que no corresponde a ninguno de los tipos especificados en el protocolo, la conexión deberá cerrarse inmediatamente por seguridad.

lenght	**id = "H"**	
4 *bytes*	1 *byte*	...
unsigned int	*string*	

Figura 5.1: Estructura general de un mensaje Pixtream

5.2.3. Identificación de los pares

Cada par tiene una identificación única dentro de la red de pares. La identificación consiste en una cadena de 20 *bytes*. Los primeros seis *bytes* de la identificación se usan para identificar el tipo y versión del cliente Pixtream de la siguiente manera: los primeros dos *bytes* contienen una identificación del cliente; la cual está compuesta por dos letras mayúsculas (A-Z), los siguientes cuatro *bytes* contienen dígitos de la versión del cliente, siendo los dos primeros la versión mayor y los siguientes la versión menor. Los siguientes 14 *bytes* de la identificación corresponden a un número hexadecimal que identifique al par. Los clientes pueden implementar cualquier estrategia para generar el número de identificación, pero se recomienda usar el estándar UUID[4] en su versión 4 (*random*).

[4] *Unique Universal Identification* es un estándar para permitir a los sistemas distribuidos obtener claves de identificación únicas [51]

Un ejemplo de identificación de un par es:

```
PX0101a2b499c312ffc5
```

5.2.4. Estado de los pares

En una red Pixtream, cada par mantiene una lista de las conexiones que tiene con otros pares. Para un determinado par, cada miembro de la lista debe tener un estado en los siguientes aspectos:

- **Bloqueado o desbloqueado**: Si un par remoto bloquea al par, significa que no responderá ninguna petición hasta que no se desbloquee.
- **Interesado o no interesado**: Si un par remoto está interesado, significa que mandará peticiones al par en cuanto tenga la oportunidad.

Cada par es responsable de llevar un registro de los estados de cada conexión. Este registro deberá tener cuatro campos con los siguientes valores:

1. `bloqueo_par_local`
2. `interes_par_local`
3. `bloqueo_par_remoto`
4. `interes_par_remoto`

5.2.5. Mensajes de la red de pares

Mensaje *Handshake*

El mensaje *Handshake* o "apretón de manos" es el primer mensaje que se envía en una conexión entre dos pares. Cuando un par A establece una conexión TCP con un par B, lo primero que debe hacer es enviar un mensaje *Handshake*. Inmediatamente después, el par B deber responder con otro mensaje *Handshake*. Si cualquiera de los dos pares recibe algún mensaje o conjunto de datos diferente de un mensaje *Handshake*, la conexión será cancelada inmediatamente.

El mensaje *Handshake* básicamente contiene información sobre la identificación de cada par y el protocolo sobre el cual se están comunicando. Cuando un par A decide establecer una conexión con un par B lo hace debido a que ha recibido la dirección y la identificación de par B a través del *rastreador*. Si el mensaje *Handshake* que devuelve el par B contiene una identificación diferente de la especificada por el *rastreador*, la conexión se debe cerrar inmediatamente.

length	**id = "H"**	**protocol id**	**extensions**	**peer id**
4 *bytes*	1 *byte*	17 *bytes*	8 *bytes*	20 *bytes*
unsigned int	*string*	*string*	*string*	*string*

Figura 5.2: Estructura del mensaje *Handshake*

Campo	**Descripción**
length	Prefijo de cuatro *bytes* con el tamaño del mensaje. En este caso siempre serán 36 *bytes*.
id = "H"	Prefijo con la identificación del mensaje.
protocol id	Cadena de identificación del protocolo. En este caso esta cadena siempre será "Pixtream Protocol". En futuras versiones del protocolo esta cadena puede cambiar si se rompe la compatibilidad hacia atrás.
extensions	Este campo de ocho *bytes* está reservado para futuras expansiones del protocolo. Por defecto este campo será ignorado por los pares, pero se recomienda llenar el espacio con pares.
peer id	Cadena de 20 *bytes* con la identificación única del par que manda el mensaje.

Tabla 5.5: Campos del mensaje *Handshake*

length	**id = "C"**
4 *bytes*	1 *byte*
unsigned int	*string*

Figura 5.3: Estructura del mensaje *Choke*

Campo	**Descripción**
length	Prefijo de cuatro *bytes* con el tamaño del mensaje. En este caso siempre será 1 *byte*.
id = "C"	Prefijo con la identificación del mensaje.

Tabla 5.6: Campos del mensaje *Choke*

Mensaje *Choke*

Este mensaje se envía para informar de que se ha bloqueado un par. Un par A envía un mensaje *Choke* a un par B para informarle de que ha sido bloqueado debido a que el par A no puede atender peticiones de más pares.

Mensaje *UnChoke*

Este mensaje funciona de manera muy similar al mensaje *Choke* de la sección 5.2.5. Se usa para informar a un para que ha sido desbloqueado y por tanto puede mandar peticiones de pedazos.

length	**id = "U"**
4 *bytes*	1 *byte*
unsigned int	*string*

Figura 5.4: Estructura del mensaje *UnChoke*

Campo	**Descripción**
length	Prefijo de cuatro *bytes* con el tamaño del mensaje. En este caso siempre será de 1 *byte*.
id = "U"	Prefijo con la identificación del mensaje.

Tabla 5.7: Campos del mensaje *UnChoke*

Mensaje *Interested*

Este mensaje lo usa un par para informar a otro que está interesado en sus pedazos.

length	**id = "I"**
4 *bytes*	1 *byte*
unsigned int	*string*

Figura 5.5: Estructura del mensaje *Interested*

Mensaje *NotInterested*

Este mensaje funciona de la misma manera que el mensaje *Interested* de la sección 5.2.5. Lo envía un par para informar a otro que ya no está interesado en sus pedazos.

Campo	Descripción
length	Prefijo de cuatro *bytes* con el tamaño del mensaje. En este caso siempre será de 1 *byte*.
id = "I"	Prefijo con la identificación del mensaje.

Tabla 5.8: Campos del mensaje *Interested*

length	**id = "N"**
4 *bytes*	1 *byte*
unsigned int	*string*

Figura 5.6: Estructura del mensaje *NotInterested*

Mensaje *RequestDataPacket*

Este mensaje se usa para solicitar a un par un pedazo del flujo multimedia. Para que un par A pueda solicitar un pedazo a un par B se deben cumplir ciertas condiciones:

1. El par B debe poseer la pieza en cuestión.
2. El par A no debe estar bloqueado por el par B. De otra forma los mensajes de petición serán ignorados.
3. El par B debió haber recibido un mensaje *Interested* previamente del par A.

length	**id = "Q"**	**sequence**
4 *bytes*	1 *byte*	4 *bytes*
unsigned int	*string*	*unsigned int*

Figura 5.7: Estructura del mensaje *RequestDataPacket*

Mensaje *CancelRequestDataPacket*

Este mensaje se envía para cancelar una petición realizada previamente con el mensaje *RequestDataPacket*. Las peticiones pueden ser canceladas por dos razones: porque el pedazo solicitado ya ha sido obtenido de otro par o porque el pedazo solicitado se ha desbordado de la ventana deslizante y por lo tanto ya no es posible reproducirlo a tiempo y será descartado.

Campo	Descripción
length	Prefijo de cuatro *bytes* con el tamaño del mensaje. En este caso siempre será de 1 *byte*.
id = "N"	Prefijo con la identificación del mensaje.

Tabla 5.9: Campos del mensaje *NotInterested*

Campo	Descripción
length	Prefijo de cuatro *bytes* con el tamaño del mensaje. En este caso siempre serán 5 *bytes*.
id = "Q"	Prefijo con la identificación del mensaje.
sequence	Identificador de secuencia del pedazo que se solicita

Tabla 5.10: Campos del mensaje *RequestDataPacket*

Mensaje *DataPacket*

Este mensaje contiene un pedazo del flujo multimedia en una sesión de *streaming* Pixtream. Para que este mensaje sea enviado, primero debió enviarse una petición usando el mensaje *RequestDataPacket*.

El par fuente es el encargado de generar los pedazos, dividiendo el flujo multimedia y asignando un identificador de secuencia y una estampa de tiempo a cada uno. El mensaje tiene longitud variable. El final del mensaje contiene el campo *payload* que varía de acuerdo con la longitud de los paquetes generados por el par fuente.

Mensaje *HeartBeat*

Este mensaje se usa para notificar a los pares que se está vivo. Los pares deben mandar "latidos de corazón" a todos los pares con quien tengan conexiones para indicar que siguen pendientes de la conexión. Si pasado un tiempo de 30 segundos un par no recibe un *HeartBeat* de otro, procederá a cancelar la conexión porque se entiende que el par está inactivo o desconectado o no puede responder. Este procedimiento se denomina *peer timeout*.

Mensaje *GotPiece*

Este mensaje se usa para indicar a los pares interesados que se ha recibido un nuevo pedazo del flujo multimedia. Cada par debe llevar un registro de qué pedazos son poseídos por qué par. Gracias a esto se pueden implementar algoritmos como *rarest first*.

length	**id = "X"**	**sequence**
4 *bytes*	1 *byte*	4 *bytes*
unsigned int	*string*	*unsigned int*

Figura 5.8: Estructura del mensaje *CancelRequestDataPacket*

Campo	**Descripción**
length	Prefijo de cuatro *bytes* con el tamaño del mensaje. En este caso siempre serán 5 *bytes*.
id = "X"	Prefijo con la identificación del mensaje.
sequence	Identificación de secuencia del pedazo cuyo solicitud va a ser cancelada.

Tabla 5.11: Campos del mensaje *CancelRequestDataPacket*

length	**id = "D"**	**sequence**	**timestamp**	**payload**
4 *bytes*	1 byte	4 *bytes*	4 *bytes*	indefinido
unsigned int	*string*	*unsigned int*	*unsigned int*	bytes

Figura 5.9: Estructura del mensaje *DataPacket*

Campo	**Descripción**
length	Prefijo de cuatro *bytes* con el tamaño del mensaje. El valor varía de acuerdo con cada mensaje
id = "D"	Prefijo con la identificación del mensaje.
sequence	Número de secuencia del pedazo.
timestamp	Estampa de tiempo generada por el par fuente cuando el pedazo fue creada. Con esto se puede llevar un control de tiempo de la reproducción del pedazo.
payload	*Bytes* en crudo con el contenido del pedazo. Este campo es de longitud variable.

Tabla 5.12: Campos del mensaje *DataPacket*

length	**id = "B"**
4 *bytes*	1 *byte*
unsigned int	*string*

Figura 5.10: Estructura del mensaje *HeartBeat*

Campo	Descripción
length	Prefijo de cuatro *bytes* con el tamaño del mensaje. En este caso siempre será 1 *byte*.
id = "B"	Prefijo con la identificación del mensaje.

Tabla 5.13: Campos del mensaje *HeartBeat*

length	**id = "G"**	**sequence**
4 *bytes*	1 *byte*	4 *bytes*
unsigned int	*string*	*unsigned int*

Figura 5.11: Estructura del mensaje *GotPiece*

Mensaje *PieceBitField*

Este mensaje se usa para informar a otros pares los pedazos que actualmente tiene un par. Este mensaje es similar a *GotPiece* (sección 5.2.5), pero sirve para indicar la posesión de más de un pedazo por mensaje.

Cuando dos pares entablan una conexión, por lo general se mandan cada uno un mensaje *PieceBitField* para informarse mutuamente de los pedazos que cada uno posee. De ahí en adelante, es deber de cada par mantener la lista de pedazos que cada par posee e informar sobre sus adquisiciones a otros pares usando el mensaje *GotPiece*.

El tamaño de este mensaje es variable.

length	**id = "F"**	**first sequence**	**last sequence**	**payload**
4 *bytes*	1 *byte*	4 *bytes*	4 *bytes*	indefinido
unsigned int	*string*	*unsigned int*	*unsigned int*	bytes

Figura 5.12: Estructura del mensaje *PieceBitField*

Mensaje *RequestPieceBitField*

Este mensaje se usa para solicitar a un par el envío de un mensaje *PieceBitField* (sección 5.2.5). Se usa cuando por algún motivo un para pierde cuenta de los pedazos que tiene otro par o cuando por alguna razón el conjunto de pares no llegó después del "apretón de manos".

Campo	Descripción
length	Prefijo de cuatro *bytes* con el tamaño del mensaje. En este caso siempre será 5 *bytes*.
id = "G"	Prefijo con la identificación del mensaje.
sequence	Secuencia del pedazo que se ha obtenido.

Tabla 5.14: Campos del mensaje *GotPiece*

Campo	Descripción
length	Prefijo de cuatro *bytes* con el tamaño del mensaje
id = "F"	Prefijo con la identificación del mensaje.
first sequence	Primer pedazo indicado por el campo de bits
last sequence	Ultimo pedazo indicado por el campo de bits
payload	Campo de bits indicando los pedazos que se posee. Se utiliza una se utiliza una serie de *bytes* para representar los pedazos que se poseen. Cada bit de la secuencia indica la posesión o no posesión de un pedazo dependiendo si el campo es uno o cero. El número de *bytes* de este campo se redondea al número de *bytes* necesarios para representar n bits donde n es el número de pedazos que se desea informar. Los bits sobrantes se deberán ignorar. Los pedazos por fuera del campo de bits serán tomados como no poseídos.

Tabla 5.15: Campos del mensaje *PieceBitField*

length	**id = "R"**
4 *bytes*	1 *byte*
unsigned int	*string*

Figura 5.13: Estructura del mensaje *RequestPieceBitField*

Campo	Descripción
length	Prefijo de cuatro *bytes* con el tamaño del mensaje. En este caso siempre será 1 *byte*.
id = "R"	Prefijo con la identificación del mensaje.

Tabla 5.16: Campos del mensaje *RequestPieceBitField*

Capítulo 6

Implementacion

Este capítulo describe cómo se desarrollo la implementación del prototipo Pixtream, teniendo en cuenta la metodología empleada, las herramientas utilizadas y los mecanismos de prueba.

6.1. Metodología de desarrollo

Para el desarrollo del prototipo de Pixtream se utilizó una metodología llamada *Extreme Hacking* propuesto por Andy Jewel [52]. *Extreme Hacking* es un subconjunto de Programación Extrema [53] que ha sido adaptado para proyectos con un sólo desarrollador y sin un cliente externo, condiciones que se cumplen en el caso de este proyecto.

Extreme Hacking propone la utilización de algunas prácticas conocidas de programación extrema:

- Pruebas unitarias.
- *Refactorización* sin piedad.
- Principio "No lo vas a necesitar" (YAGNI).
- Metáfora del sistema.
- Lanzamientos frecuentes-
- Estándares de codificación.

6.2. Herramientas

Esta sección menciona las herramientas utilizadas para la elaboración del prototipo de Pixtream

6.2.1. *Frameworks* P2P

Antes de comenzar la implementación del prototipo de Pixtream, primero se analizó la posibilidad de utilizar algún *framework* o biblioteca de programación para el desarrollo de aplicaciones P2P. Dentro de la exploración que se hizo se encontraron tres alternativas:

1. JXTA [54]
2. Brunet[1]
3. Vertex[2]

Aunque estos *frameworks* presentan varias características que facilitan el desarrollo de una aplicación P2P, también imponen restricciones al desarrollo de la misma, haciéndolos poco adecuados para el prototipo de Pixtream. Un ejemplo de estas restricciones tiene que ver con el protocolo utilizado para la comunicación. Mientras que proyectos como JXTA y Brunet utilizan un protocolo propio basado en XML, diseñado para potenciar la descentralización y ampliar las posibilidades de comunicación, Pixtream utiliza un protocolo basado en datos binarios cuya prioridad es la eficiencia. Encapsular el protocolo Pixtream en el de otro *framework* significaría una sobrecarga considerable en las necesidades de comunicación del programa.

En el caso de Vertex, se realizaron algunas pruebas sencillas pero se encontró que el proyecto está muy inestable y además abandonado. Debido a esto se decidió no utilizar ningún *framework* P2P y diseñar Pixtream desde la base.

6.2.2. Plataforma de desarrollo

Para el desarrollo de la aplicación se escogió el lenguaje de programación Python[3] en su versión 2.6. Python es un lenguaje dinámico bastante ágil y muy recomendado para la elaboración de prototipos.

Existen tres razones principales por la cuales se escogió Python para implementar el prototipo de Pixtream:

1. Suficiente experiencia del desarrollador. Lo cual acorta considerablemente el tiempo de desarrollo
2. Disponibilidad de herramientas necesarias, en este caso un *framework* para *networking* asíncrono.
3. Utilización previa en otros proyectos P2P. Proyectos como Bittorrent, el servidor de *streaming* Flumotion[4], o incluso CoolStreaming han sido implementados utilizando Python. Esto no sólo da confianza en la plataforma, sino que también permite explorar código fuente real pertinente al caso.

[1] `http://boykin.acis.ufl.edu/wiki/index.php/Brunet`
[2] `http://divmod.org/trac/wiki/DivmodVertex`
[3] `http://www.python.org`
[4] `http://www.flumotion.com`

Adicionalmente se usó el *framework* Twisted[5] para *networking* asíncrono. Este *framework* permitió desarrollar las conexiones del protocolo Pixtream basándose en un patrón de reactor que facilitó en gran medida el desarrollo del prototipo.

6.2.3. Otras herramientas

Para el control de versión se utilizó Mercurial[6], un sistema de control de versiones distribuido utilizado por varios proyectos de software libre. Gracias a Mercurial fue posible llevar un control completo de todos los cambios en el código fuente. En total se realizaron cerca de 200 revisiones. Las revisiones completas están detalladas en el anexo C.

Para las pruebas unitarias se usó el módulo estándar PyUnit de Python.

6.3. Pruebas

Las pruebas que se hicieron a Pixtream se pueden separar en dos categorías: Pruebas de software y pruebas de desempeño. Las pruebas de software tienen como objetivo validar que el programa funcione como debería funcionar. Las pruebas de desempeño, descritas en el capítulo 7, sirven para determinar qué tan efectivas son las técnicas usadas en el diseño del prototipo.

Dentro de las pruebas de software se destacan dos categorías: Pruebas unitarias y pruebas manuales.

6.3.1. Pruebas unitarias

Las pruebas unitarias son pruebas automáticas que son escritas en el mismo lenguaje que la aplicación y que pueden ejecutarse cuacntas veces sea necesario. Las pruebas unitarias ayudan a verificar la corrección de funciones y métodos dentro del programa.

En total se hizo una batería de cerca de 50 pruebas unitarias. Utilizando los sistemas de prueba automática de PyUnit, es posible ejecutarlas automáticamente cada vez que sea necesario.

Las prebas se encuentran en el directorio *test* del código fuente de la aplicación.

6.3.2. Pruebas manuales

Desafortunadamente, la complejidad de una aplicación P2P hace que las pruebas unitarias se queden cortas a la hora de verificar la validez del software. En Pixtream se encontraron muchos errores que no se pudieron detectar con pruebas automáticas. Por esta razón también se preparó un sistema de pruebas manuales que permite crear varias instancias de los pares P2P, en diferentes escenarios y probar que todo funciona correctamente.

[5] `http://www.twistedmatrix.com`
[6] `http://selenic.com/mercurial/`

Para poder verificar los datos, teniendo en cuenta el comportamiento asíncrono del sistema, fue necesario crear un sistema de bitácoras o *logs* para poder hacer análisis *post-mortem* de los fallos encontrados.

Las pruebas manuales consisten en una serie de programas *scripts* que lanzan automáticamente un determinado número de pares y servidores multimedia y permiten realizar sesiones de *streaming* automáticas que ayudan a detectar errores.

Las pruebas manuales se encuentran en el directorio *manualtests* del código fuente de la aplicación.

Capítulo 7

Experimentos y resultados

En el capítulo 3 se estudiaron distintas propuestas de sistemas de *media streaming* P2P. Como se había mencionado antes, la mayoría de estas propuestas evalúan su rendimiento utilizando bases teóricas y simulaciones. Uno de los objetivos de este proyecto es contrastar los resultados teóricos de las propuestas analizadas con los resultados del comportamiento de una implementación de Pixtream en un ambiente real.

Para poder realizar la comparación de dichas propuestas contra Pixtream, primero es necesario conocer cuales fueron los experimentos realizados y los resultados obtenidos por cada propuesta. Posteriormente se diseña un experimento para probar el rendimiento de Pixtream basándose en los experimentos de las propuestas. Finalmente se extraen los resultados y se comparan.

7.1. Resultados de las simulaciones de las propuestas analizadas

Las propuestas analizadas en el capítulo 3 fueron: BitTorrent, GNUStream, BEAM, AnySee y PULSE. A continuación se muestra por cada propuesta los parámetros analizados en las comparaciones, los escenarios utilizados y los resultados obtenidos.

7.1.1. BitTorrent

Las características de la propuesta para modificar BitTorrent para adaptarlo al *media streaming* fueron descritas en la sección 3.1 (página 12).

Configuración del experimento

Para probar las modificaciones hechas al protocolo BitTorrent por Shah y Pâris [9], se creo una simulación de una red con 100 pares. Se consideró un entorno homogéneo en el que todos pares tienen la misma capacidad de subida.

También se asumió que el enjambre se comporta correctamente y que todos los pares continúan aportando ancho de banda de salida aún después de haber terminado la transmisión.

Para simplificar el modelo, se ignoraron algunas complejidades de las dinámicas del protocolo TCP. Se asumieron conexiones idealizadas que no se pierden y que presentan iguales tasas de transferencia por conexión.

La configuración usada para el experimento fue:

Tamaño del video	150 MB
Tamaño de los pedazos	256 KB
Número inicial de *seeds*	1
Ancho de banda	10 Mbps
Máximo número de conexiones concurrentes	5
Número de pares retornados por el rastreador	50

Tabla 7.1: Configuración de las simulaciones de BitTorrent.

Parámetros analizados

Para el experimento se utilizaron dos métricas:

- **Tasa de éxito**: esta métrica representa la continuidad en la reproducción del flujo multimedia. Se mide tomando en cuenta la cantidad de pedazos que llegan a un par antes de que se cumpla el momento límite para la reproducción.
- **Rendimiento de la red**: esta métrica representa la utilización de la red en cuanto a ancho de banda. Se mide tomando la cantidad total de *bytes* transmitidos con respecto a la capacidad de ancho de banda de la red.

Escenarios previstos

Los experimentos se realizan comparando las modificaciones hechas al protocolo BitTorrent para adaptarlo a *media streaming*. Las comparaciones fueron hechas en los siguientes escenarios:

- **Con selección de pedazos**: en donde se utiliza la modificación en el algoritmo de selección de pedazos, usando una ventana deslizante.
- **Con tit-for-tat**: en donde se utiliza la modificación en el algoritmo de selección de pares, usando el algoritmo *tit-for-tat* aleatorio al inicio.

Resultados obtenidos

En las pruebas realizadas por la propuesta de Shah y Pâris, utilizando el algoritmo de selección de pedazos basado en una ventana deslizante y *rarest-first*,

se observó una tasa de éxito del 83.3 % contra una tasa de éxito del 8.1 % usando un algoritmo secuencial y 23.6 % usando la política original de BitTorrent. Además, probaron el efecto del tamaño de la ventana deslizante con un rango de tamaño desde 90 hasta 540 pedazos. Los resultados mostraron un mejor rendimiento con una ventana deslizante de 120 pedazos.

También se evaluó la política de selección de pares. Se comparó el modelo aleatorio adicionando *tit-for-tat* contra el modelo tradicional de BitTorrent *tit-for-tat*. En las pruebas realizadas condiciones de ancho de banda se mostró una aumento del 20 % en promedio en el rendimiento de la red.

7.1.2. GNUStream

Las características de la propuesta GNUStream, se describen en la sección 3.2 (página 14).

Configuración del experimento

El documento de la propuesta de GNUStream [27] menciona un experimento realizado con la siguiente configuración:

Resolución del vídeo	352x240
Cuadros por segundo	30
Tasa de reproducción	143 KB/s
Número de pares	4
Número de pares fuente	3
Ancho de banda de salida por par	60 KB/s

Tabla 7.2: Configuración de las simulaciones de GNUStream.

Parámetros analizados

El experimento analizó los siguientes aspectos:

- **Tamaño del *buffer* requerido**: que es el tiempo en el que el par receptor tiene que guardar los datos en el *buffer* para poder obtener una reproducción continua del vídeo.
- **Detección de fallas y recuperación**: que se refiere a cómo reacciona la red frente a fallas de los pares.

Resultados obtenidos

En el experimento se comprobó que el retraso adecuado para una transmisión continua, usando tres pares fuente y un par receptor, era de 11.2 segundos. Además, al desconectar uno de los pares y agregar uno nuevo, el tiempo que demoró el receptor en encontrar una nueva fuente fue de un segundo, y la recuperación del tamaño del *buffer* duró 26 segundos.

7.1.3. BEAM

Las características de la propuesta BEAM, se describen en la sección 3.3 (página 16).

Configuración del experimento

Duración del vídeo:	1 hora
Tasa de reproducción:	512 Kb/s
Tamaño de los pedazos:	16 Kb
Número de pares:	1024

Tabla 7.3: Configuración de las simulaciones de BEAM.

Parámetros analizados

Los experimentos realizados con BEAM examinan:

- **Latencia promedio**: el tiempo de diferencia en la reproducción del vídeo en la fuente y en cada cliente
- **Falla en la entrega**: el número de paquetes que llegan a un par después que se cumpla el tiempo de reproducción.
- **Escalabilidad y robustez**: qué tal se comporta el sistema ante los fallos.

Escenarios previstos

Para el experimento se plantearon dos posibles escenarios:

- **Sistema estable**: un sistema sin fallas, donde todos los nodos permanecen conectados hasta el final de la transmisión.
- **Sistema inestable**: un sistema donde varios pares abandonan la sesión en la mitad de la transferencia.

Resultados obtenidos

El experimento mostró un índice de falla en la entrega del 0.38 %, la latencia promedio fue de 8 segundos. Además de eso, se probó la efectividad del uso del factor de utilidad para determinar el grado de colaboración de cada par, mostrando que el 99.82 % de los pares tuvieron una latencia menor que los pares con menor factor de utilidad.

En las pruebas donde las fallas fueron agregadas, se probó el caso en el que 50 % de los pares activos fallaran en la mitad de la reproducción. Con este cambio el índice de falla aumentó un 1.2 %. La utilización del ancho de banda en todos los casos fue del 82 %.

7.1.4. AnySee

Las características de la propuesta AnySee, se describen en la sección 3.4 (página 17).

En el caso de AnySee, que usa una topología de múltiples árboles y un funcionamiento general bastante diferente de todos los demás, los autores realizaron un experimento basado en múltiples redes superpuestas y simulaciones de topologías lógicas y físicas.

Configuración del experimento

Para el experimento se consideraron dos tipos de topologías: lógica y física. Se generaron trazas de 2000 nodos utilizando un generador de topología de red. También se realizaron utilizando varias redes superpuestas para la transmisión del contenido multimedia.

Duración del vídeo:	1800 segundos
Tasa de reproducción:	300 Kb/s
Tamaños de las redes superpuestas:	1, 4, 8, 12
Número de conexiones por nodo:	1 a 40
Ancho de banda por conexión:	250 kb/s

Tabla 7.4: Configuración de las simulaciones de AnySee.

Parámetros analizados

- **Utilización de recursos**: Esto es la tasa del ancho de banda usado entre las conexiones usadas y todas las conexiones
- **Índice de continuidad**: El porcentaje del flujo multimedia que llega al cliente a tiempo.

Escenarios previstos

Para la simulación se probaron dos escenarios:

- **Entorno estable**: En este escenario todos los nodos se unen a la red al inicio de la transmisión y se van al final de esta.
- **Entorno inestable**: En este escenario los nodos se van y se unen a la red en forma aleatoria.

Resultados obtenidos

En el entorno estable, los resultados mostraron un índice de continuidad entre 0.98 y 1.0. Los resultados muestran además que el índice de continuidad es más cercano a 1.0 utilizando más redes superpuestas.

Adicionalmente, la utilización de recursos mejora considerablemente al aumentar el número de redes superpuestas utilizadas en la sesión. Resultando en un índice de 0.3 con dos redes superpuestas hasta 0.60 con 12.

En el entorno inestable el índice de continuidad varió mucho más, pero se mantuvo relativamente estable con índices entre 0.9 y 0.98. Para el caso de la utilización de recursos, los índices se mantuvieron en rangos similares a las pruebas realizadas en el entorno estable.

7.1.5. PULSE

Las características de la propuesta PULSE, se describen en la sección 3.5 (página 18).

Configuración del experimento

Para los experimentos de PULSE, se utilizó un simulador de red simple con enlaces con ancho de banda ajustable. La documentación disponible no da detalles concretos de la configuración del experimento, pero se menciona el hecho de que se usaron diferentes tamaños de red, distribución de ancho de banda y parámetros de *buffer*. El tamaño de la ventana deslizante fue fijado en 64 pedazos.

Parámetros analizados

El único parámetro analizado en las pruebas simuladas de esta propuesta es la latencia. Definida como el tiempo de espera que se requiere en cada nodo para empezar a reproducir el contenido.

Escenarios previstos

Se probaron dos escenarios:

- **Nodos simultáneos**: en este caso todos los nodos ingresan a la red uniformemente al inicio de la simulación.
- **Nodos aleatorios**: en este caso los nodos ingresan a la red aleatoriamente después del segundo 120.

Resultados obtenidos

Para el primer escenario se obtuvo una inestabilidad inicial en la red con un pico en la latencia de 120 pedazos del flujo multimedia. A medida que la red se va estabilizando, la latencia disminuye hasta un promedio de 20 pedazos.

En el segundo escenario, igualmente se presenta una inestabilidad al inicio de la transferencia, con un pico de 200 pedazos de latencia. Sin embargo, a medida que transcurre la sesión, la red es considerablemente más inestable que en el primer escenario. Presentando una latencia promedio de 25 pedazos, con picos que van hasta los 100.

7.1.6. Coolstreaming/DONet

Las características de la propuesta Coolstreaming, se describen en la sección 3.6 (página 20).

Configuración del experimento

Duración del video	120 Minutos
Tasa de transferencia	500 Kb/s
Duración de los pedazos	1 Segundo
Tamaño de la ventana deslizante	60 Segmentos
Tiempo límite de reproducción	10 segundos
Número de nodos probados	10, 50, 100, 150 y 200

Tabla 7.5: Configuración de las simulaciones de PULSE.

Parámetros analizados

- **Índice de continuidad**: definido por el porcentaje de pedazos que llegan a un nodo antes de que se cumpla el tiempo límite de reproducción.
- **Índice de escalabilidad**: que se refiere al porcentaje de ancho de banda que tiene que utilizarse para mandar mensajes relacionados con el control de la red.

Escenarios previstos

Al igual que los experimentos realizados por otras propuestas, se toman dos escenarios diferentes para las pruebas:

- **Escenario estático**: donde todos los nodos arribando al mismo tiempo al inicio de la transferencia. En este caso se da una espera de 1 minuto para que todos los nodos entren a la red y luego se inicia la transferencia.
- **Escenario dinámico**: donde se analiza el caso de nodos que abandonan la red o fallan. En este caso cada nodo tiene un patrón aleatorio *on/off*, en el cual se conectan y se desconectan en un cierto lapso.

Resultados obtenidos

Los resultados obtenidos de CoolStreaming fueron los siguientes:

Para el escenario estático, el índice de escalabilidad se mantuvo por debajo de 1.8 %. Dependiendo del número de pares con los que se compartan pedazos del flujo multimedia, el índice de escalabilidad aumenta linealmente. Por ejemplo para el caso de dos pares el índice se mantiene en 0.6 % para redes de 10, 50, 100,

150 y 200 nodos. Para 3 pares es 0.8 %, para 4 pares 1 % y así sucesivamente de forma lineal.

Las pruebas realizadas al índice de continuidad varían desde 95 % hasta 97 %. Los resultados más bajos se obtienen cuando los pares se configuran para usar pocos compañeros y se estabiliza cerca del 97 % con más de 4 compañeros.

En el entorno dinámico el índice de escalabilidad se mantuvo cerca de 1.8 % , para el caso del índice de continuidad, se puede apreciar un detrimento en este con el entorno dinámico que está directamente asociado con el tiempo en el el que los pares están ausentes. En el caso en el que la intermitencia es más frecuente el índice de continuidad se ubica cerca del 88 %, mientas que con periodos más largos se acerca a 94 %.

7.2. Descripción de los experimentos

7.2.1. Introducción

Después de que el prototipo de Pixtream fue diseñado e implementado con base en las características observadas en otros trabajos, la siguiente fase en el proyecto es realizar varias pruebas experimentales al prototipo para poder comparar qué tan bien se desempeña en comparación con los experimentos simulados realizados por varias propuestas.

Es importante aclarar que estas pruebas son diferentes a las realizadas en la sección 6.3 durante la implementación del prototipo. El objetivo de estas pruebas no es encontrar fallos en la construcción del software sino más bien medir la efectividad de los algoritmos y técnicas usados.

7.2.2. Datos medidos

Con base en los experimentos descritos en las propuestas de sistemas de *media streaming* P2P, se seleccionaron algunas características del comportamiento de Pixtream que deben ser medidos en los experimentos realizados en este proyecto. Al medir estos mismos datos en el comportamiento de Pixtream, es posible después realizar una comparación con la información que se plantea en los documentos de las propuestas como resultados de las simulaciones. Esto permite realizar un contraste entre las simulaciones y los datos del sistema real.

La información que se se necesita medir es: índice de continuidad, rendimiento de la red, distribución de los recursos, latencia promedio y tolerancia a fallos. A continuación se describe en qué consiste cada uno de ellos.

Índice de continuidad

Uno de los aspectos más importantes en una sesión de *streaming*, es que la reproducción sea lo más continua posible y sin interrupciones. Para medir la calidad de la transmisión en estos términos, se examina el índice de continuidad de la sesión. Este índice se define por el número de paquetes que llegan a tiempo antes del tiempo límite de la reproducción. Así, por ejemplo, en un índice

de continuidad de 98 % se puede decir que 98 de cada cien pedazos del flujo multimedia llegaron a tiempo.

Utilización de la red

El rendimiento de la red se refiere a la tasa de transferencia usada durante la sesión de *streaming* con respecto a la capacidad estimada total de los enlaces. Sí un nodo hace transporta $100Kb/s$ en una red con capacidad de $1000Kb/s$, entonces podemos decir que la utilización del enlace es del 10 %

Latencia promedio

La latencia se refiera al tiempo que pasa desde que el contenido multimedia es generado hasta que es reproducido por el cliente final. Para medir la latencia en Pixtream se tiene que tomar el tiempo desde que el par fuente envía un pedazo hasta que un par normal lo recibe y lo envía al reproductor multimedia.

Tolerancia a fallos

La tolerancia a fallos se refiere a la capacidad que tiene la red para recuperarse de pérdidas de nodos. Esto puede pasar porque las conexiones fallan o simplemente porque los pares se retiran voluntariamente. Para medir la tolerancia a fallos se examinan las variaciones en latencia, índice de continuidad y rendimiento de la red en casos en los que los nodos se retiran abruptamente.

7.2.3. Metodología de medición

Los experimentos se realizaron en una sala de cómputo donde se utilizaron hasta 16 computadores simultáneamente.

Para poder realizar las mediciones en Pixtream, se realizaron algunas modificaciones al programa para que lleve un registro continuo de todas las actividades de un par. De esta forma, cada programa par lleva un archivo bitácora o *log* de todas las operaciones que realiza en la sesión: desde la inicialización de la red, la conexión al rastreador, la conexión a otros pares y todos los mensajes que son recibidos o enviados. Cada entrada del *log* contiene el momento exacto en que los hechos ocurrieron. Gracias a este mecanismo es posible medir, con buena precisión, datos como la latencia y el índice de continuidad.

El desarrollo de cada experimento consiste en una sesión de *streaming* con Pixtream utilizando un par fuente y n pares normales. Cada par normal deberá correr en un computador individual, sin embargo, en algunos casos se usarán varios pares por computador con el objetivo de probar redes más grandes. En estos casos se aplicará un sistema de control de ancho de banda en la interfaz *loopback* de los computadores para restringir los recursos al promedio de la red. Es decir que dos pares que estén corriendo sobre el mismo computador se comunicarán con un ancho de banda similar a aquel disponible para la comunicación con otros pares que estén en otros computadores.

Por cada par se realizará una medición del ancho de banda utilizado por la aplicación y se activará el *log* para registrar todos los acontecimientos internos.

Medición del índice de continuidad

La medición del índice de continuidad se realiza examinando los *logs* de cada par. Con el deslizamiento de la ventana en cada par, se registra en el *log* aquellos paquetes que fueron enviados al reproductor multimedia y aquellos que no pudieron ser enviados debido a que no se obtuvieron a tiempo. El índice de continuidad se mide como se describe en la fórmula 7.1

$$I_c = \frac{P_c}{P_t} \qquad\qquad P_t = P_c + P_f \tag{7.1}$$

Donde:

I_c : Índice de continuidad
P_c : Número de pedazos que llegaron a tiempo
P_f : Número de pedazos que no llegaron a tiempo
P_t : Número total de pedazos

En el caso de los escenarios de control, en los que se realzó experimentos a sistemas de *streaming* tradicional, la medición del índice de continuidad fue un poco diferente. Para este caso se contabilizaron las paradas que hace el reproductor multimedia cliente para recargar el *buffer* de la transmisión. Utilizando este dato se puede tomar la relación entre el número de paradas y el tiempo total del archivo, pudiendo medir la latencia. La fórmula 7.2 describe cómo se calculó el índice de continuidad en este caso.

$$I_c = \frac{t_r}{t_b} \tag{7.2}$$

Donde:

t_r : Tiempo total del archivo
t_b : Tiempo en que el reproductor carga buffer

Medición de la latencia

Para medir la latencia, se compara el tiempo de envío al reproductor multimedia de cada pedazo, con respecto al tiempo en el que el servidor de streaming envió el mismo paquete al par fuente. Se toma un promedio por todos los pedazos recibidos durante una sesión y se obtiene la latencia promedio por par.

Adicionalmente se puede tomar la latencia total por sesión promediando cada par.

La fórmula 7.3 describe cómo se mide la latencia promedio en las pruebas realizadas a Pixtream.

$$L_i = \frac{t_p - t_f}{M} \qquad\qquad L_s = \frac{\sum_{i=0}^{N_p} L_i}{P} \tag{7.3}$$

Donde:

L_i : Latencia por par
L_s : Latencia por sesión
t_p : Tiempo en el que el paquete sale del par normal
t_f : Tiempo en el que el paquete llega al par fuente
M : Número de pedazos recibidos por el par
P : Número de pares en la sesión

Medición de la utilización de la red

Para medir el uso de la red, se utiliza un programa que indica cuanto ancho de banda está usando la aplicación, en este caso el prototipo de Pixtream, y se relaciona con la capacidad total estimada del enlace utilizado.

La fórmula 7.4 describe cómo se mide la utilización de la red por par.

$$R = \frac{B_i}{B_e} \tag{7.4}$$

Donde:

R : Rendimiento de la red
B_i : Tasa de transferencia del par
B_e : Tasa de transferencia máxima estimada

Medición de la tolerancia a fallos

La tolerancia a fallos se midió comparando el comportamiento del prototipo en escenarios estables e inestables. Para los casos estables, se toma una red P2P en la cual todos los clientes entran a la red simultaneamente y permanencen en esta durante toda la sesión de *streaming*. Para el caso inestable se toma una red en la cual la mitad de los clientes abandonan la sesión en un determinado punto.

7.2.4. Herramientas utilizadas

La tabla 7.6 muestra una lista y descripción de las herramientas de software utilizadas durante los experimentos.

Herramienta	Descripción
Pixtream 1.0.1	Versión de Pixtream que se probó en los experimentos
Python Portable 2.6.4 `http://www.portablepython.com/`	Entorno de ejecución requerido para correr la versión de prueba de Pixtream. Se utilizo la versión portable para tener la posibilidad de ejecutar el programa en cualquier máquina sin necesidad de realizar instalaciones
Twisted 10.0 `http://twistedmatrix.com/`	Librería de redes requerida por Pixtream.
VLC Portable 1.0.3 `http://www.videolan.org/`	Reproductor multimedia utilizado. Adicionalmente se utilizó como servidor de *streaming*. Igualmente se usó la versión portable para evitar instalaciones.
iftop 0.17 `http://www.ex-parrot.com/pdw/iftop/`	Monitor de ancho de banda por aplicación e interfaz para Linux. Utilizado para medir el uso de ancho de banda en el caso de que se usen varios pares por computador.
bmon 2.0.1 `http://freshmeat.net/projects/bmon/`	Monitor de ancho de banda para Linux utilizado para medir el ancho de banda de bajada y de subida en el par fuente.
DU Metter 1.0 `http://www.dumeter.com/`	Monitor de ancho de banda para Windows usando para medir el ancho de banda usado por los pares.
trickle 1.0.6 `http://monkey.org/~marius/trickle/`	Programa para restringir el uso de ancho de banda de un programa en Linux. Utilizado en los experimentos para restringir el ancho de banda en los casos necesarios.

Tabla 7.6: Herramientas utilizadas en los experimentos

7.2.5. Material multimedia

Los experimentos se realizaron con dos tipos de material multimedia. Por un lado se utilizó un archivo de audio ligero y con pocos requerimientos de ancho de banda. Adicionalmente, se utilizó también un archivo de vídeo con audio mucho más grande y con altos requerimientos de ancho de banda. Los dos tipos de archivo sirvieron para medir las capacidades de Pixtream en diferentes condiciones. Las tablas 7.7 y 7.8 describen las especificaciones técnicas de ambos archivos.

Duración del audio	2 Minutos 26 segundos
Tasa de transferencia	166 kbps
Frecuencia de muestreo	48000 Hz
Codec	OGG VORBIS
Duración de los pedazos	4 KiB
Tamaño de la ventana deslizante	60 Segmentos
Tiempo límite de reproducción	5 segundos

Tabla 7.7: Especificaciones técnicas del archivo de audio probado en los experimentos.

Duración del vídeo	42 minutos:57 segundos
Dimensiones	624 x 352
Tasa de fotogramas	24 fps
Tasa de bits	1112 kbps
Frecuencia de muestreo del audio	48000 Hz
Tasa de bits del audio	103 Kbps
Códec Video	XViD MPEG-4
Códec Audio	MPEG 1 Audio, Layer 3 (MP3)
Duración de los pedazos	4 KiB
Tamaño de la ventana deslizante	60 Segmentos
Tiempo límite de reproducción	10 segundos

Tabla 7.8: Especificaciones técnicas del archivo de vídeo probado en los experimentos.

7.2.6. Escenarios probados

Los experimentos fueron realizados dentro del marco de tres escenarios diferentes: un sistema de *streaming* tradicional, un sistema Pixtream P2P estable y un sistema inestable.

Sistema de *streaming* tradicional

Con el objetivo de tomar una muestra de control, se tomó el caso de un sistema tradicional de *streaming* basado en una arquitectura cliente/servidor. Para este caso se utilizó el sistema VLC[1] en dónde se midió la continuidad de la transmisión y el uso de ancho de banda de salida contrastado con el número de clientes conectados simultáneamente. Este escenario sirve como caso de referencia para comparar el desempeño de un sistema P2P de *streaming*.

Escenario estable

En el que se probó el prototipo de Pixtream, funcionando en un ambiente en el que todos los nodos ingresaron a la red al mismo tiempo y permanecieron conectados a la red de forma estable durante toda la sesión de *streaming*.

Escenario inestable

El escenario inestable permite probar la tolerancia a fallos que presenta Pixtream. Comparando estos resultados con aquellos obtenidos en el escenario estable, es posible determinar qué tanto influye la salida de nodos en la calidad de la transmisión.

7.2.7. Experimentos realizados

En total se realizaron 32 experimentos para probar Pixtream. Como se mencionó anteriormente, los experimentos fueron realizados en tres diferentes escenarios: Sistema tradicional, P2P Estable y P2P Inestable. En los experimentos realizados se midieron los parámetros descritos en la sección 7.2.2, sin embargo, no en todos los casos se pudieron medir todos los parámetros debido a dificultadoes técnicas. En los experimentos también se probaron los dos tipos de contenido usados como materiales descritos en la sección 7.2.5.

En todos los experimentos se utilizó, además de los equipos clientes, un servidor multimedia que sirvió sólo para ese propósito. En este servidor se ejecutó el servidor de *streaming* y el par fuente en el caso de las pruebas P2P.

La tabla 7.9 (página 80) resume los 32 experimentos realizados especificando el número de clientes que se usó, el uso de equipos y el número clientes por equipo. Adicionalmente se muestra cuales materiales multimedia fueron usado así como los datos que se midieron.

7.3. Resultados obtenidos

En esta sección se muestran los resultados de las mediciones realizadas en los experimentos descritos en la sección 7.2. Las tablas y gráficos que describen los datos obtenidos de las mediciones se pueden consultar al final del capítulo.

[1] Video Lan Player `http://www.videolan.org/`

		Componentes			Material		Datos medidos		
N°	Escenario	NC	NE	CPE	AUD	VID	LAT	IC	UDR
1	S.Tadicional	1	2	1	✓	✓	✗	✓	✓
2		2	3	1	✓	✓	✗	✓	✓
3		3	4	1	✓	✓	✗	✓	✓
4		4	5	1	✓	✓	✗	✓	✓
5		6	7	1	✓	✓	✗	✓	✓
6		8	9	1	✓	✓	✗	✓	✓
7		11	12	1	✓	✓	✗	✓	✓
8		13	14	1	✓	✓	✓	✓	✓
9		16	17	1	✓	✓	✓	✓	✓
10		32	17	2	✓	✗	✗	✓	✓
11		64	17	4	✓	✗	✗	✓	✓
12	P2P Estable	1	2	1	✓	✓	✗	✓	✓
13		2	3	1	✓	✓	✗	✓	✓
14		3	4	1	✓	✓	✗	✓	✓
15		4	5	1	✓	✓	✗	✓	✓
16		6	7	1	✓	✓	✗	✓	✓
17		8	9	1	✓	✓	✗	✓	✓
18		11	12	1	✓	✓	✗	✓	✓
19		13	14	1	✓	✓	✗	✓	✓
20		16	17	1	✓	✓	✓	✓	✓
21		32	17	2	✓	✓	✓	✓	✓
22		64	17	4	✓	✓	✓	✓	✓
23	P2P Inestable	1	2	1	✓	✓	✗	✓	✓
24		2	3	1	✓	✓	✗	✓	✓
25		3	4	1	✓	✓	✗	✓	✓
26		4	5	1	✓	✓	✗	✓	✓
27		6	7	1	✓	✓	✗	✓	✓
28		8	9	1	✓	✓	✗	✓	✓
29		11	12	1	✓	✓	✗	✓	✓
30		13	14	1	✓	✓	✗	✓	✓
31		16	17	1	✓	✓	✓	✓	✓
32		32	17	2	✓	✓	✓	✓	✓

Tabla 7.9: Resumen de los experimentos realizados en Pixtream
NC: Número de clientes, NE: Número de equipos, CPE: Clientes por equipo, AUD: Prueba de audio, VID: Prueba de vídeo, LAT: Latencia, IC: Índice de continuidad, UDR: Uso de la red

7.3.1. Uso de la red

Las mediciones al uso de la red se realizaron en los tres escenarios descritos anteriormente y utilizando los dos tipos de material descritos en la sección 7.2.5.

Los resultados de las mediciones utilizando el material ligero de audio se muestran en la tabla 7.11 y son ilustrados en la figura 7.1 (página 87).

Para el caso del archivo de vídeo más pesado, los resultados se muestran en la tabla 7.12 y se ilustran en la figura 7.2 (página 88).

7.3.2. Índice de continuidad

El índice de continuidad se midió usando la metodología descrita en la sección 7.2.2.

La tabla 7.13 (página 89) muestra los resultados obtenidos de las mediciones del índice de continuidad en el archivo de audio. La figura 7.3 ilustra los valores de las medidas obtenidas.

Para el caso de las pruebas realizadas con el material de vídeo, la tabla 7.14 y la figura 7.4 (página 90) ilustran los resultados obtenidos.

7.3.3. Latencia

La latencia fue medida en Pixtream calculando la latencia promedio de cada pedazo del flujo multimedia en cada minuto de la transferencia. Las observaciones se hicieron tomando en cuenta sólo el archivo de audio ligero y en el caso del escenario de *streaming* tradicional sólo se tomó el caso 13 y 16 clientes simultáneos. Para este último caso sólo se registró la latencia inicial de cada sesión. La tabla 7.15 (página 91) muestra los resultados de las mediciones de latencia en distintos escenarios y con diferentes números de clientes. La figura 7.5 (página 92) ilustra la tabla.

7.4. Discusión y análisis de los resultados

En la sección 7.3 se mostraron los resultados de las mediciones realizadas en el prototipo de Pixtream que se desarrollo. Además, se midió también el rendimiento de un sistema *streaming* tradicional basado en una arquitectura cliente/servidor. En esta sección se discuten y explican esos resultados teniendo en cuenta los distintos parámetros que fueron medidos: uso de red, índice de continuidad, latencia y tolerancia a fallas.

7.4.1. Comparación del uso de red

Las primeras pruebas que se realizaron en el prototipo fueron las del uso de la red. Al observar los resultados en las pruebas tanto de audio como de vídeo, se puede notar una primera clara diferencia entre la arquitectura P2P empleada en Pixtream y la arquitectura cliente servidor usada en VLC.

En las pruebas realizadas al servidor tradicional (VLC), se nota un incremento casi lineal en el uso de la red a medida que la cantidad de clientes aumenta. Sin embargo, en la medida en que la capacidad máxima del enlace de red es alcanzada, se nota como el uso de la red cesa su crecimiento y toca techo. Este comportamiento se puede apreciar en la figura 7.1 (página 87), donde el crecimiento es lineal hasta con 48 nodos simultáneos y de ahí en adelante el crecimiento se ve acotado. En este caso, el crecimiento lineal es sostenido con muchos nodos debido a que el archivo utilizado para las pruebas es ligero y eso hace que el ancho de banda necesario sea mucho menor. Es mucho más fácil observar el efecto *techo* en la figura 7.2 (página 88) debido a que se uso un archivo de vídeo mucho más exigente en términos de ancho de banda. En este caso se puede observar que el techo se alcanza cerca de los 13 nodos. De ahí para adelante el crecimiento cesa y el uso de red empieza a ser constante. Como se verá más adelante este fenómeno implica serios problemas en la continuidad de la transmisión.

En los resultados de las mediciones de uso de la red en el prototipo Pixtream se nota un comportamiento diferente al del sistema tradicional. En primer lugar, el ascenso en el uso de ancho de banda en el par fuente no es lineal como en el caso de VLC. Las mediciones para las pruebas con distinto número de nodos son más inestables y es difícil identificar algún patrón que las relacione. No obstante, algo que se puede observar claramente es que, en los casos del sistema P2P, la máxima capacidad de no es alcanzada nunca. Si bien el uso de la red se mantiene alto siempre cuando el número de clientes crece, el uso de la red no se dispara; al contrario, tiende a estabilizarse. La razón de esto tiene que ver con el hecho de que en el sistema P2P todos los nodos están contribuyendo al envío de paquetes multimedia, lo cual alivia la carga en el par fuente. Esto contrasta con el caso del servidor tradicional en dónde sólo el servidor central se encarga de enviar paquetes y todos los clientes actúan como receptores únicamente.

7.4.2. Continuidad en la reproducción

En las mediciones del índice de continuidad es donde más claramente se puede observar las ventajas del sistema P2P Pixtream frente al sistema tradicional de *streaming*. Como se discutió en la sección anterior, en la medida que el número de clientes crece en el sistema tradicional, el uso de red alcanza el tope permitido por el enlace y ya no puede subir más. Este factor influye directamente en el índice de continuidad.

En la figura 7.3 (página 89), se puede observar como mientras en número de clientes es bajo, el índice de continuidad en el caso de VLC es bastante alto; muy cercano al 100 %. Sin embargo, cuando los clientes superan los 32, la continuidad se ve claramente afectada y con tendencia a la baja. Este hecho contrasta fuertemente con las pruebas realizadas en el prototipo P2P Pixtream, en dónde la continuidad se mantiene muy cerca del 100 % sin importar el número de clientes.

En el caso en el que se usó un material más exigente en cuanto a ancho de banda, el descenso en el rendimiento del sistema tradicional es mucho más

notorio. En la figura 7.4 (página 7.4), se observa como el índice de continuidad en el caso de VLC inicia muy bien pero en cuanto aumenta el número de nodos, rápidamente cae a niveles muy bajos. En el caso del sistema P2P Pixtream el caso es contrario: el inicio no es muy bueno pero luego se nota una estabilización muy cercana al 100 % de continuidad.

Es en estas mediciones en las que se nota más evidentemente el problema que tienen los sistemas de *media streaming* tradicionales en cuanto a escalabilidad. El cuello de botella en el ancho de banda de salida es determinante para el número de clientes que puede servir con buena calidad. Es también aquí donde se nota la ventaja del enfoque P2P para *media streaming*, ya que las sesiones se vuelven escalables junto con el número de clientes aprovechando su ancho de banda de salida.

Un comportamiento bastante interesante, el cual se puede notar especialmente en el caso de las pruebas en vídeo, es que, para el caso de Pixtream P2P, el índice de continuidad presenta una tendencia al alza a medida que el número de clientes. Esto se interpreta como que la calidad de la transmisión multimedia aumenta a medida que la red es más grande; contrastando completamente con lo que sucede en el caso del servidor tradicional. También se puede observar que la brecha entre la calidad de la transmisión entre el escenario estable y el inestable tiende a disminuir. Esto significaría también que el sistema se vuelve más tolerante a fallas y más estable a medida que crece.

7.4.3. Latencia

Un aspecto en el que claramente el sistema P2P tiene desventaja es en la latencia. En los experimentos realizados, el sistema de control cliente/servidor siempre consiguió una latencia considerablemente más baja que en las pruebas realizadas con Pixtream. Esto se debe principalmente a que los datos viajan directamente desde el servidor hasta el cliente, mientras que en una red P2P los paquetes deben ser divididos, distribuidos y reconstruidos. La figura figura 7.5 (página 7.5) ilustra los resultados de latencia en diferentes escenarios. Mientras que los resultados de las pruebas en el sistema cliente/servidor muestran una latencia ligeramente superior a dos segundos, en Pixtream oscila entre 4 y 10 segundos. Como es natural suponer, las redes Pixtream más grandes presentaron una mayor latencia.

7.4.4. Tolerancia a fallos

La tolerancia a los fallos es algo difícil de medir. También es difícil comparar la tolerancia a fallos con un sistema de *streaming* tradicional debido a que este último no tiene problemas en el caso de desconexión en clientes. En los experimentos realizados la tolerancia a fallos se midió teniendo en cuenta los comportamientos en los escenarios estable e inestable.

La diferencia más significativa se puede observar en las mediciones de índice de continuidad, en las que las pruebas hechas en un escenario estable tuvieron

un mejor rendimiento que aquellas hechas en entornos inestables. En la latencia también se notó un cierto impacto, aunque menor dado que no es tan grave.

Otro hecho que es interesante notar es que, tanto en el escenario de un sistema P2P estable como inestable, el uso de la red es bastante similar y es difícil notar una tendencia hacia uno u otro en cuanto a uso de red se refiere. Esto probablemente indique que ante una eventual caída de nodos fuente, la tendencia no necesariamente es sobrecargar el par fuente, sino que las peticiones se distribuyen sobre toda la red.

7.5. Comparación de los resultados

Luego de realizar los experimentos para medir el rendimiento y comportamiento de Pixtream, la siguiente parte consiste en comparar aquellos resultados obtenidos con los que se había planteado en las simulaciones presentadas en el capítulo 3. Esta constituye una de las partes más importantes de este proyecto debido a que dicha comparación es uno de los objetivos principales. Si bien la gama de trabajos relacionados con sistemas de *streaming* P2P es amplia, la mayoría de estos sólo han quedado en planteamiento teórico y pruebas en simuladores. El mayor aporte del proyecto Pixtream consiste en proveer una implementación real de un sistema de *streaming* P2P que pueda ser probado en ambientes reales y permita contrastar los resultados de simulaciones.

En la sección 7.1 se resumieron los experimentos y simulaciones realizados por cada trabajo estudiado en este proyecto. Debido a que no todos los trabajos probaron las mismas cosas ni midieron los mismos parámetros, es necesario extraer sólo aquellas mediciones que equivalen a las hechas en Pixtream. En la tabla 7.10 se puede ver un resumen de los resultados obtenidos por los trabajos estudiados; mostrando sólo aquellos que tienen relevancia para una comparación con Pixtream.

Trabajo	Medición	Resultados	
Bittorrent	Continuidad	83 %	
BEAM	Continuidad	96 %	
BEAM	Latencia	8 segundos	
GNUStream	Latencia	11 segundos	
PULSE	Latencia	20 - 120 pedazos	
PULSE	Continuidad	95 % - 97 % estable	88 % - 94 % inestable

Tabla 7.10: Resumen de los resultados expuestos por los trabajos estudiados relevantes para la comparación con Pixtream.

De la tabla 7.10 se pueden observar dos parámetros que pueden ser comparados directamente con los resultados obtenidos en Pixtream: latencia e índice de continuidad. Adicionalmente, teniendo en cuenta que, al igual que en Pixtream, en algunos casos las mediciones de índice de continuidad se hicieron tanto en

escenarios estables como inestables, también se puede realizar una comparación en cuanto a tolerancia a fallos.

7.5.1. Comparación de la continuidad de la transmisión

De todos los experimentos simulados realizados en los trabajos estudiados, el de medición del índice de continuidad de la transmisión es el que más pruebas tiene. Es en este aspecto que se tienen más datos para realizar la comparación con Pixtream y también es uno de los aspectos más importantes de la investigación.

Según los resultados de las simulaciones realizadas en los trabajos que se estudiaron, el índice de continuidad varía desde 83 % en Bittorrent hasta 98 % en PULSE. En el caso de las pruebas realizadas a Pixtream, las variaciones van desde 97 % hasta 99 % en el caso de audio ligero y desde 92 % hasta 100 % en el caso de vídeo más pesado. En ambos casos se habla de escenarios estables.

Teniendo en cuenta que los resultados de los distintos trabajos no son equivalentes, ya que se hicieron en condiciones diferentes, de todo modos es posible decir que los resultados obtenidos de los experimentos en Pixtream están a rangos similares a aquellos planteados por las simulaciones. El resultado es más parecido si se le compara con pruebas más parecidas como las que se hicieron en BEAM y PULSE. En estos casos Pixtream tuvo nivel inferior más pequeño, pero en general se mantuvo dentro de los rangos mostrados en las propuestas. El rango inferior es posiblemente debido a inestabilidades en el entorno real en el que se probó Pixtream, en contraste con las simulaciones en entornos ideales.

Para los casos en los que se midieron también los índices de continuidad en entornos inestables, se nota un caso ligeramente diferente. Para este caso comparamos solamente el caso de PULSE que plantea resultados en escenarios estables e inestables (AnySee también experimenta en los dos escenarios, pero los resultados se omiten debido a que se usa una topología diferente). Para el caso de audio ligero, los resultados son sorprendentemente similares: Pixtream 89 % - 95 % frente a 88 % - 94 % en PULSE. No obstante, en el caso de vídeo más pesado los resultados son bastante diferentes. En Pixtream se tiene un rango de 79 % - 100 % con un promedio cercano a los 85 %. Esto muestra que el rendimiento de Pixtream en un entorno real fue inferior al registrado por PULSE en las simulaciones. La inestabilidad de la red en el entorno real es la principal causa de este hecho, viéndose con más notoriedad cuando se realiza una mayor exigencia a la red.

7.5.2. Comparación de la latencia

A diferencia de los resultados para el índice de continuidad, las mediciones de latencia realizadas en el prototipo de Pixtream difieren de aquellas planteadas por las simulaciones en otros trabajos. En el caso de BEAM y GNUStream, la latencia fue de ocho y once segundos respectivamente. En Pixtream, si bien la latencia comienza en un rango muy parecido al obtenido de las simulaciones

de estos dos trabajos, la latencia presenta una tendencia notable hacia abajo y luego se estabiliza entre los cuatro y seis segundos.

Las diferencias en este caso podrían explicarse por diferencias en la forma en como se midieron los datos en cada caso. Desafortunadamente es imposible dar un juicio definitivo al respecto debido a que la información proporcionada en la documentación de los trabajos estudiados no detalla la metodología exacta de los experimentos. Si la latencia fue tomada al inicio de la sesión de *streaming*, los datos coincidirían con lo que se observó en Pixtream. De otra forma no es posible crear una relación entre ambas mediciones.

Más allá de las dificultades comparativas en este caso, es destacable el hecho de que la Latencia general en Pixtream sea baja comparada con la presentada por otras propuestas. Indicando esto efectividad en los algoritmos y técnicas utilizadas para la compartición de pedazos en el sistema.

7.5.3. Comparación de la tolerancia a fallos

Según la información proporcionada por la documentación de los trabajos estudiados, una comparación en cuanto a tolerancia a fallos sólo se puede hacer con PULSE, debido a que es el único trabajo que registra los resultados obtenidos tanto en entorno estable como en entorno inestable.

Mientras que en el escenario estable el rango de continuidad de la transmisión en PULSE varía en un rango de 95 % - 97 %, en el escenario inestable varía entre 88 % y 94 %. En las pruebas con audio ligero en Pixtream la variación va de 97 % - 99 % en el entorno estable y 89 % - 95 % en el entorno inestable. En las pruebas a vídeo pesado la variación es de 92 % hasta 100 % en entorno estable a 79 % - 100 % en entorno inestable.

Comparando los resultados de PULSE y Pixtream, se puede concluir que la tolerancia a fallos es menor en Pixtream. Esto debido a que al presentarse inestabilidad en la red, retirando algunos clientes, al sistema le cuesta más trabajo recuperarse. Los clientes que dependían de los ausentes tardan demasiado tiempo en encontrar nuevas fuentes de pedazos. Una muy posible causa de este problema se debe a que en una red real es más difícil detectar la ausencia de clientes de manera instantánea. Este procedimiento es más efectivo en una simulación y por ende se justifica la diferencia de resultados.

N° de clientes	S. tradicional	S. P2P estable	S. P2P inestable
0	393	472	316
1	22197	20693	20505
2	41760	44620	42616
3	67455	56859	64338
4	85060	88880	87245
6	129383	130369	130183
8	152220	148659	179941
11	209989	245357	230335
13	253163	580716	640316
24	538312	571094	436497
32	611656	553123	501911
48	735268	416621	372592
64	735104	458995	615220

Tabla 7.11: Resultados de las mediciones de uso de la red en audio.
Los valores están en *bytes* por segundo (B/s)

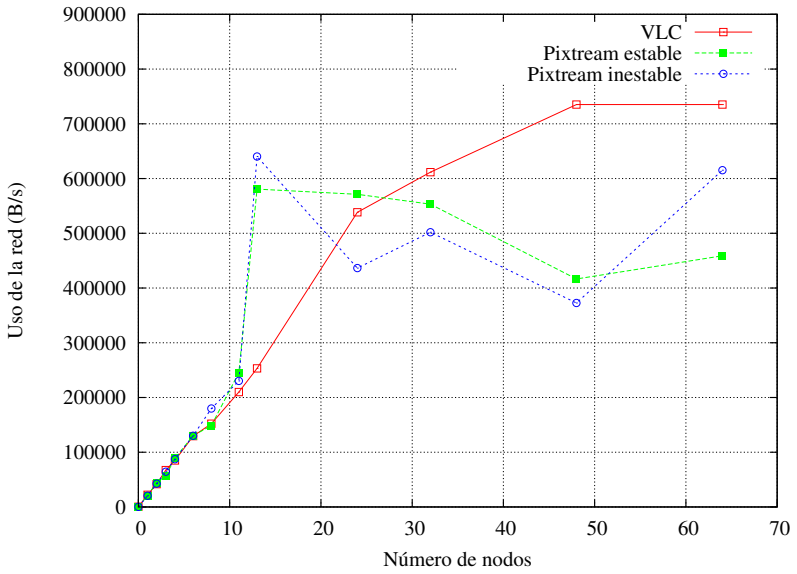


Figura 7.1: Uso de la red en distintos escenarios para audio

N° de clientes	S. tradicional	S. P2P estable	S. P2P inestable
0	168	200	200
1	85337	79546	84000
2	203984	198000	218000
3	363924	303450	273450
4	401231	401231	281231
6	565030	421231	321231
8	645182	353137	553137
11	732309	573564	473564
13	742341	551231	501231
16	761131	523564	593564
24	761343	521234	431234
32	761824	555887	505887
48	764598	495887	565887
64	767034	530086	510086

Tabla 7.12: Resultados de las mediciones de uso de la red en vídeo.
Los valores están en *bytes* por segundo (B/s)

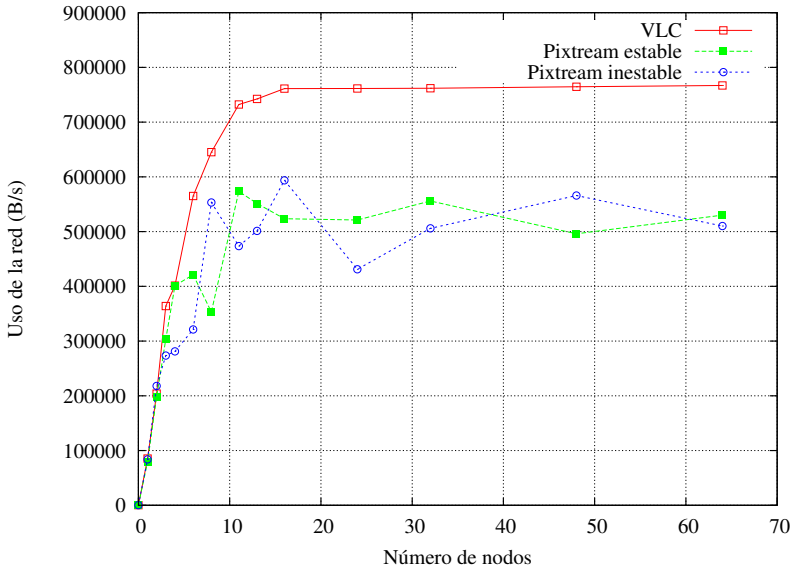


Figura 7.2: Uso de la red en distintos escenarios para vídeo

N° de clientes	S. tradicional	S. P2P estable	S. P2P inestable
1	100	98.89	92.82
2	100	97.78	94.23
3	97	98.09	92.55
4	98	97.55	93.36
6	98	99.46	91.66
8	97	99.71	93.32
11	98	98.74	95.08
13	98	98.21	89.02
24	97	97.63	94.26
32	90	98.63	92.07
48	84	99.05	93.89
64	73	98.29	90.52

Tabla 7.13: Resultados de las mediciones del índice de continuidad para audio

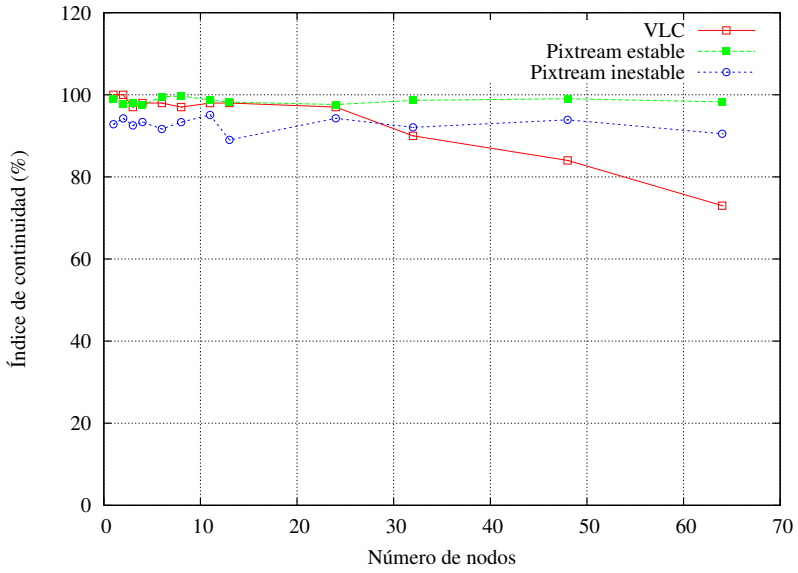


Figura 7.3: Índice de continuidad en distintos escenarios para audio

N° de clientes	S. tradicional	S. P2P estable	S. P2P inestable
1	100	100.0	100.0
2	100	89.2	90.0
3	97	92.4	85.1
4	97	95.6	82.4
5	95	96.1	79.1
6	90	98.0	80.1
7	83	97.2	85.1
8	72	95.9	88.2
11	56	97.4	87.0
13	40	99.1	91.2
16	35	97.6	85.0
32	20	98.2	82.9
64	18	97.3	89.0

Tabla 7.14: Resultados de las mediciones del índice de continuidad para vídeo

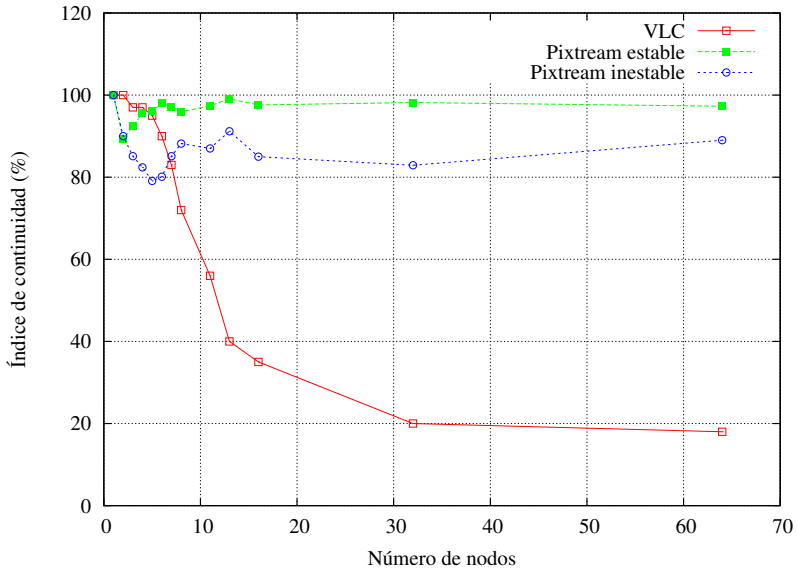


Figura 7.4: Índice de continuidad en distintos escenarios para vídeo

Tiempo (min)	P2P estable 16 nodos	P2P inestable 16 nodos	P2P estable 32 nodos	P2P inestable 32 nodos	P2P estable 64 nodos
1	9.2	9.2	9.8	9.9	10.3
2	8.2	8.5	8.8	9.3	9.6
3	8.3	8.5	8.2	9.2	10
4	7.4	8.4	7.8	8.7	9.2
5	6.6	7.2	8	8.2	8.1
6	6.3	6.9	7	7.1	8.4
7	6	6.4	6.9	6.5	7.7
8	5.5	6.5	5.9	6.5	7.1
9	5	5.2	5.6	5.9	6.2
10	4.4	4.9	4.9	5.2	5.8
11	4.8	4.9	5.3	5.8	5.8
12	4.5	5.4	4.9	5.4	6.4
13	4.1	4.5	4.6	5.8	6
14	4.9	5.4	4.6	5.6	5.8
15	4.1	5.5	4.9	5.9	5.9
16	4.1	5.4	5.5	5.3	6.5
17	4.9	5.5	5.5	6	6
18	4.8	5	4.7	5.6	5.6
19	4.1	5.3	4.5	5.7	5.9
20	4.2	5.4	4.8	5.1	5.7
21	4	5	4.9	5.7	6.3
22	4.8	5.2	4.5	5.1	5.7
23	4.8	4.8	5.4	5.1	6.1
24	4.9	5.1	5.5	5.7	6.3

Tabla 7.15: Resultados de las mediciones de latencia en segundos por tiempo transcurrido

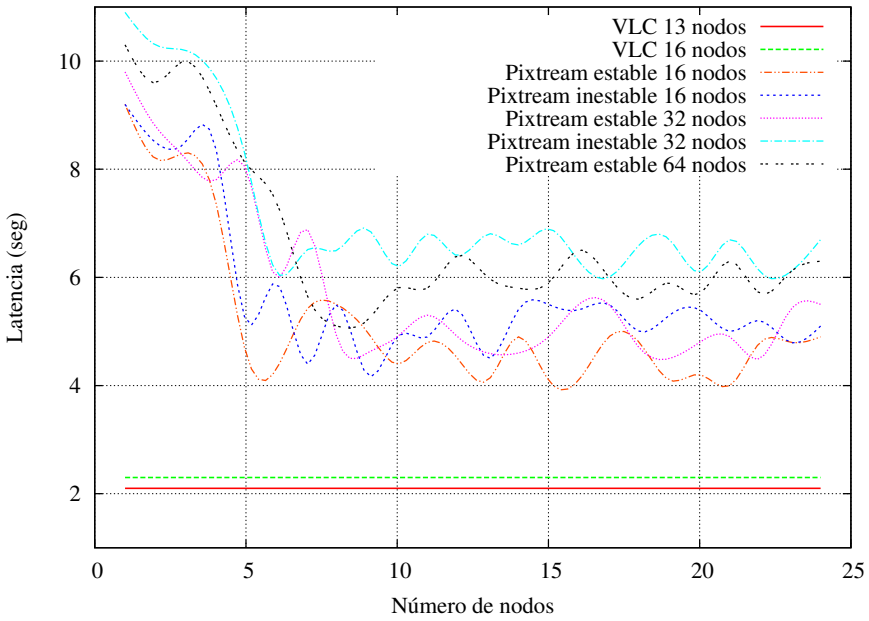


Figura 7.5: Latencia en distintos escenarios

Capítulo 8

Problemas, conclusiones y trabajos futuros

8.1. Dificultades y soluciones

Durante el desarrollo de este proyecto se encontraron varias dificultades y se plantearon algunas soluciones a ellas:

8.1.1. La falta de especificaciones completas

Al momento de estudiar los trabajos previos en el tema de *media streaming*, si bien se encontraron varias propuestas sobre el tema, la documentación disponible sobre ellas no siempre estuvo completa. Esto dificultó en gran medida aspectos importantes de este proyecto: el desarrollo del protocolo Pixtream y la comparación de resultados de simulaciones con resultados reales.

En el caso del protocolo, no fue posible encontrar documentación de ninguna de las propuestas estudiadas sobre un protocolo completo de *streaming* P2P. Esto dificultó bastante el proceso de diseño del protocolo en Pixtream, aunque también se constituyo en un aporte importante del proyecto. La solución para este problema fue basarse en otros protocolos P2P generales, que no aplican al tema de *streaming*, pero que sirven como base. La especificación se basó en documentos de proyectos como Bittorrent y Gnutella.

En el aspecto de los resultados de las simulaciones, el problema fue más grave porque se comprometía el cumplimiento de uno de los objetivos del proyecto. En muchos casos se encontró que la documentación sobre los experimentos y pruebas realizados en otros trabajos fue bastante insuficiente. Muchos detalles faltaron, como por ejemplo, la metodología exacta de los experimentos y los parámetros empleados. La solución aquí fue usar sólo los datos que se tenían, aunque se redujera en cierta forma el rango de comparación. En muchos casos no se pudo hacer una comparación directa, pero al menos se hizo una aproximación.

8.1.2. Pruebas y la depuración

Durante la construcción del prototipo, hubo una dificultad grande en el proceso de realizar pruebas. Desarrollar una aplicación P2P en general es una tarea difícil porque las pruebas no dependen de una sola instancia del programa sino que requiere construir una red completa que funciona de manera asíncrona. En muchos casos, el uso de métodos de pruebas tradicionales se queda corto y aparecieron muchos errores difíciles de encontrar y corregir.

Para solucionar este problema se tuvo que hacer uso de herramientas propias para realizar pruebas. Se construyó un sistema de bitácora o *log* que permitiera saber qué está pasando en cada instancia de la aplicación en todo momento. También se elaboraron programas que permitieran la ejecución automática de múltiples instancias de la aplicación en diferentes condiciones.

8.1.3. La separación de la codificación multimedia

En la etapa de elaboración de la arquitectura, se decidió que Pixtream sólo debería operar en la etapa de transporte de un sistema de *streaming*. Debido a las limitaciones en el alcance del proyecto, se evitó trabajar en el área de codificación y decodificación multimedia, prefiriendo el uso de herramientas externas para este trabajo. Esto tuvo la ventaja de hacer viable la construcción del prototipo, pero también generó limitaciones en el mismo.

Al hacer que Pixtream desconociera la codificación multimedia, fue imposible hacer que la información del contenido pudiera ser obtenida y recodificada durante la transmisión multimedia. En otras palabras, lo que esto significa es que durante una sesión de *streaming* en Pixtream no es posible obtener información solamente de una parte del contenido y por ende los clientes no pueden unirse en la mitad de la sesión.

Una solución a esto durante el lapso del proyecto no pudo alcanzarse. Modificaciones en este aspecto requerirían un nuevo proyecto y por eso son propuestas como trabajo futuro en la sección 8.3

8.2. Conclusiones

Durante la realización del proyecto se pudo concluir lo siguiente:

- Se estudiaron seis trabajos que proponen sistemas de *media streaming* basados en redes P2P. En todos los trabajos se encontraron características similares en aspectos como: topología, sistema de entrada, estrategias de selección y controlo de continuidad.

- Basándose en las características comunes encontradas en trabajos anteriores sobre el tema de *media streaming*, fue posible diseñar una arquitectura de software de un sistema de *streaming* P2P.

- Pese a la falta de ejemplos documentados de protocolos de *streaming* P2P, fue posible diseñar uno durante el proyecto. El protocolo se basó completamente en las características estudiadas de propuestas sobre el tema y en la arquitectura que se diseño. Adicionalmente, se encontraron características importantes que debe tener un protocolo de este estilo, como mensajes ágiles de poco tamaño que permitan una comunicación rápida.

- El protocolo desarrollado constituye un aporte importante para el área de *streaming* P2P. El estudio del estado del arte de los trabajos sobre el tema mostró que no hay protocolos documentados para está área. El hecho de haber especificado un protocolo en Pixtream puede facilitar en gran medida la elaboración de futuros trabajos en el mismo campo.

- Se elaboró un prototipo de software de un sistema de *streaming* P2P que puedo ser probado en entornos reales. El prototipo se convierte en la primera implementación del protocolo Pixtream. Adicionalmente se convierte en un escalón sobre el cual se puede trabajar para probar futuros algoritmos y técnicas utilizadas en el campo del *streaming* P2P. Este prototipo fue liberado como software libre.

- Los experimentos realizados con el prototipo desarrollado muestran las ventajas y desventajas que tiene el enfoque *peer-to-peer* en el campo de *media streaming*. Se pudo observar que la continuidad en la transmisión no se ve afectada con el aumento de clientes, haciendo el sistema mucho más escalable en comparación con el clásico esquema cliente/servidor. Por otro lado se observaron desventajas como un aumento en la latencia y una dependencia de los clientes que puede generar desmejoras en la calidad del *streaming* en el momento en que algunos clientes abandonan la red

- En algunos casos se pudo contrastar los resultados que otros trabajos han planteado, y que habían sido producto de simulaciones y proyecciones teóricas, con resultados reales de observaciones realizadas en un prototipo de Pixtream ejecutándose sobre una red real. La comparación mostró que los resultados simulados son bastante similares a la realidad en varios casos como en la continuidad de la transmisión, mientras que en la tolerancia a fallos y latencia difieren ligeramente.

- En otros casos fue imposible realizar una comparación directa entre los resultados de otros trabajos y los experimentos en Pixtream. Esto se debió principalmente a la falta de información sobre cómo se realizaron las simulaciones en otros trabajos y la poca especificación de los algoritmos usados en algunas propuestas.

8.3. Trabajos futuros

El tema de *media streaming* P2P es muy grande. Este proyecto se queda corto para todas las posibilidad de investigación y desarrollo que hay en esta

área. A continuación se mencionan algunos posibles trabajos futuros que podrían complementar lo realizado en Pixtream.

8.3.1. Integración de codificación multimedia

Como se mencionó en la sección 8.1.3, para muchos casos prácticos hace falta que el sistema de *streaming* P2P tenga la capacidad de entender la codificación del material multimedia que está siendo transmitido. Esto permitiría, entre otras cosas, acceso a pares en medio de una sesión de *streaming* y una división más inteligente del flujo multimedia. Un interesante trabajo futuro podría ser el soporte de codecs como H.264 o Theora en Pixtream.

8.3.2. Nuevas técnicas de distribución

El uso de nuevas estrategias de división del flujo multimedia también podría potenciar lo realizado en Pixtream. Un trabajo futuro muy valioso podría ser la integración de un sistema como MDC (*Multiple Description Coding*) [42] en remplazo de la división secuencial del flujo multimedia que se usa en Pixtream. MDC permite dividir un contenido multimedia en capas. Cada capa permite reproducir el contenido y entre más capas se tengan mejor es la calidad. De esta forma la pérdida de paquetes significaría menor calidad sin perder información.

8.3.3. Aplicar esquemas de descentralización

Dentro del área de redes P2P puras, un trabajo futuro importante sería aplicar esquemas de descentralización a Pixtream. La estrategia sería reemplazar el rastreador por un sistema distribuido que no requiera ninguna entidad central. Un ejemplo de esto son las tablas de *hash* distribuidas o DHT. Las DHT ya se están empezando a usar en sistemas con entidades centralizadas como Bittorrent y la misma idea podría aplicarse a Pixtream.

8.3.4. Búsqueda y publicación de contenido en P2P

Los sistemas de publicación y descubrimiento de contenido en Pixtream son bastante simples. Un buen trabajo futuro puede ser la aplicación de estos sistemas. Se puede agregar mecanismos de búsqueda y publicación, así como también permitir la posibilidad de participación a los usuarios aportando opiniones a la manera de una red social.

8.3.5. Ampliación de la capacidad de conectividad

Uno de los grandes problemas en toda red *peer to peer* es la dificultad que tiene algunos participantes de actuar como servidores. Esto se debe a que, últimamente, es común encontrar clientes que participan en internet desde una red privada utilizando traducción de direcciones (NAT) o *proxies*. Existen varias técnicas para solucionar estos problemas y un buen trabajo futuro podría ser ampliar las capacidades de Pixtream en esta área.

8.3.6. Exploración de nuevas tecnologías WEB

Gracias a estándares como HTML5, la web está cambiando y se está volviendo más poderosa. Uno de los estándares definidos dentro de HTML5 son los WebSockets. Utilizando esta herramienta podría ser posible implementar un cliente P2P en la web. Un trabajo futuro asombroso podría ser portar Pixtream a la web utilizando las nuevas tecnologías que hay disponibles.

Bibliografía

[1] S. Zehnder, "Live P2P Video Streaming Framework," *Diploma Thesis, Communications System Group. University of Zürich*, 2008.

[2] M. Topic, *Streaming Media Demystified*. McGraw-Hill Professional, 2002.

[3] E. Setton and B. Girod, *Peer-to-Peer Video Streaming*. Springer, 2007.

[4] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: an application level multicast infrastructure," pp. 5–5, 2001.

[5] R. Schollmeier, "A Definition of Peer-to-Peer Networking towards a Delimitation Against Classical Client-Server Concepts," *Proceedings of WATM-Eunice*, 2001.

[6] G. Wen, H. Longshe, and F. Qiang, "Recent Advances in Peer-to-Peer Media Streaming Systems," *China Communications*, p. 52, 2006.

[7] G. Diestro and J. García, "Difusion Multimedia Sobre Internet," *Ampliación en Redes. Universidad de Valladolid*, 2002.

[8] E. Stream2Stream, "Documentación de Stream2Stream." `http://s2s.sourceforge.net/about.php`.

[9] P. Shah and J. F. Pâris, "Peer-to-Peer Multimedia Streaming Using BitTorrent," pp. 340–347, 2006.

[10] N. Magharei and R. Rejaie, "Understanding mesh-based peer-to-peer streaming," *Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*, 2006.

[11] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches," pp. 1424–1432, 2007.

[12] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming," *Proceedings of IEEE INFOCOM*, 2007.

[13] D. Carra, R. L. Cigno, and E. W. Biersack, "Graph Based Analysis of Mesh Overlay Streaming Systems," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 9, pp. 1667–1677, 2007.

[14] F. Pianese, J. Keller, and E. W. Biersack, "PULSE, a Flexible P2P Live Streaming System," 2006.

[15] G. Marfia, G. Pau, P. Di Rico, and M. Gerla, "P2P Streaming Systems: A Survey and Experiments," *STreaming Day 2007*, 2007.

[16] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over peers," *Stanford InfoLab*, 2002.

[17] D. A. Tran, K. A. Hua, and T. Do, "ZIGZAG: an efficient peer-to-peer scheme for media streaming," *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, vol. 2, 2002.

[18] S. Banerjee and B. Bhattacharjee, "Analysis of the NICE Application Layer Multicast Protocol," *Network*, vol. 4, p. 3, 2002.

[19] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: high-bandwidth multicast in cooperative environments," *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 298–313, 2003.

[20] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1456–1471, 2002.

[21] C. M. University, "End System Multicast." `http://esm.cs.cmu.edu/`.

[22] N. Magharei and R. Rejaie, "Adaptive receiver-driven streaming from multiple senders," *ACM Multimedia Systems*, vol. 11, no. 6, pp. 550–567, 2006.

[23] V. Padmanabhan and K. Sripanidkulchai, "The case for cooperative networking," 2002.

[24] V. Padmanabhan, H. J. Wang, and P. A. Chou, "Resilient peer-to-peer streaming," *11th IEEE International Conference on Network Protocols, 2003. Proceedings.*, pp. 16–27, 2003.

[25] X. Zhang, J. Liu, B. Li, and T. S. P. Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming," vol. 3, pp. 13–17, 2005.

[26] D. Purandare and R. Guha, "BEAM: An Efficient Peer to Peer Media Streaming Framework," *Proceedings 2006 31st IEEE Conference on Local Computer Networks*, pp. 513–514, 2006.

[27] D. Xu, X. Jiang, Y. Dong, and B. Bhargava, "GnuStream: a P2P media streaming system prototype," *Proceedings of 2003 International Conference on Multimedia and Expo, 2003. ICME'03.*, vol. 2, 2003.

[28] M. Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella Network," vol. 101, 2001.

[29] L. Xiaofei, J. Hai, L. Yunhao, *et al.*, "AnySee: Peer-to-Peer live streaming," *Barcelona, Spain: Proceedings of IEEE INFO-COM*, 2006.

[30] B. Cohen, "The BitTorrent Protocol Specification." `http://www.bittorrent.org`.

[31] D. Erman, D. Ilie, and A. Popescu, "Bittorrent session characteristics and models," *Traffic Engineering, Performance Evaluation Studies and Tools for Heterogeneous Networks*, vol. 61, no. 84, p. 61, 2004.

[32] H. Schulze and K. Mochalski, "Ipoque Internet Study 2008/2009." `http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009`.

[33] R. Palacios and V. Delgado, "Introducción a la Criptografía: tipos de algoritmos," *Canales de mecánica y electricidad*, 2006.

[34] T. Sundsted, "The practice of peer-to-peer computing: Introduction and history," *http://www-106.ibm.com/developerworks/java/library/j-p2p*, 2001.

[35] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the gnutella network," *IEEE Internet Computing*, pp. 50–57, 2002.

[36] Y. Liu, X. Liu, L. Xiao, L. M. Ni, and X. Zhang, "Location-aware topology matching in P2P systems," *IEEE INFOCOM*, vol. 4, pp. 2220–2230, 2004.

[37] D. Mills, "Network Time Protocol (Version 3) specification, implementation and analysis," tech. rep., RFC 1305, 1992.

[38] K. Jenkins, K. Hopkinson, and K. Birman, "A gossip protocol for subgroup multicast," *International Workshop on Applied Reliable Group Communication. IEEE Computer Society*, 2001.

[39] CoolStreaming, "Sitio Web de CoolStreaming." `http://www.coolstreaming.us`.

[40] A. J. Ganesh, A. M. Kermarrec, and L. Massoulié, "SCAMP: Peer-to-peer lightweight membership service for large-scale group communication," *Lecture notes in computer science*, pp. 44–55, 2001.

[41] P. Backx, T. Wauters, B. Dhoedt, and P. Demeester, "A comparison of peer-to-peer architectures," *Eurescom Summit*, vol. 2002, 2002.

[42] V. K. Goyal, "Multiple description coding: Compression meets the network," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 74–93, 2001.

[43] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. ProQuest Information and Learning Company, 2003.

[44] R. M. Stallman, *Free Software, Free Society: Selected Essays of Richard M. Stallman*. O'Reilly Media, Inc., 2002.

[45] D. S. team, *Cracking the Code - Peer-to-Peer Application Development*. Dreamtech Press, 2002.

[46] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC 2616: Hypertext transfer protocol–HTTP/1.1, June 1999," *Status: Standards Track*.

[47] T. Berners-Lee, R. Fielding, and L. Masinter, "RFC 3986: Uniform resource identifier (uri): Generic syntax," *The Internet Society*, 2005.

[48] T. Berners-Lee, L. Masinter, and M. McCahill, "RFC 1738: Uniform resource locators (URL), December 1994," *See Reference [138]*.

[49] D. Crockford, "RFC 4627: The application/json media type for javascript object notation (json)," *Online, accessed*, vol. 17, 2006.

[50] B. Blanc and B. Maaraoui, "Endianness or Where is Byte 0," *White Paper*, 2005.

[51] P. Leach, M. Mealling, and R. Salz, "RFC 4122: a Universally Unique IDentifier (UUID) URN Namespace," *Retrieved from http://www. ietf. org/rfc/rfc4122. txt*, 2005.

[52] A. Jewel, "Extreme Hacking," *Cunningham, Inc.*

[53] K. Beck, *Extreme programming explained: embrace change*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[54] J. D. Gradecki, *Mastering Jxta: Building Java Peer-to-Peer Applications*. John Wiley Inc. New York, NY, USA, 2002.