Universidad Rey Juan Carlos

INGENIERÍA INFORMÁTICA

Curso académico 2007-2008

Proyecto Fin de Carrera

ARETUSA: SISTEMA DE REPUTACIÓN PARA BITTORRENT

Autor: Jaime Pérez Crespo

Tutor: Antonio Fernández Anta

Tutor: Luis López Fernández

Tutor: Diego R. López García

Agradecimientos

A mi familia, amigos y compañeros por la ayuda y apoyo prestados durante la larga elaboración de este proyecto. A Antonio, Diego y Luis por comprometerse desde un principio para que pudiese llevar a cabo este proyecto. A Álvaro, Macías, Óscar, Chuso y Tomás por sus comentarios y su inestimable ayuda.

Resumen

Este trabajo presenta y estudia el problema del *free-riding*, o la obtención de recursos sin ofrecer nada en compensación, en redes de pares y más concretamente en la red *BitTorrent*. Pese a la difícil solución de dicho problema, se propone una forma de reducir su impacto e incluso solventarlo en casos concretos que impliquen un intercambio de información por encima de un cierto umbral.

En este documento se analiza el problema en toda su extensión y se realiza un estudio sobre sus posibles soluciones y su viabilidad, seguido por la descripción técnica de la arquitectura propuesta como solución y su implementación, llamada «aretusa», así como, para finalizar, una evaluación de los objetivos planteados en un inicio tras la ejecución del proyecto.

Abstract

This project introduces and studies the problem of *free-riding* (obtaining resources without sharing anything with other peers), common in all peer-to-peer networks, and more precisely in the *BitTorrent* network. Although there's no easy way to solve it, we present a method to reduce its impact or even remove it when the amount of data exchanged exceeds some threshold.

This document presents a complete analysis of the problem and studies the possible solutions and their feasibility, followed by the technical description of the suggested architecture and its implementation, called «aretusa», just as, to end up, the evaluation of the goals of the project after its final execution.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Las redes de telecomunicaciones y más concretamente Internet forman hoy parte indisoluble de nuestras vidas. De forma inconsciente, poco a poco cada uno de los dispositivos que se han ido haciendo hueco en nuestras tareas cotidianas se conectan entre ellos en una red global, una red de redes. El aumento progresivo de aparatos que se conectan a la red requiere así mismo un aumento de la capacidad de ésta. No sólo eso, sino que nosotros mismos hemos dado un salto en cuanto a las exigencias que le planteamos a nuestra conexión. Ya no basta con acceder a la red, ahora queremos acceder con unos parámetros mínimos de velocidad y calidad. Nuestro uso de la red ha cambiado, y con él la red en sí misma. Internet no es la misma que hace apenas una década.

El aumento en el nivel de exigencia se ha traducido en un aumento de prestaciones. De las «tarifas planas» que hace apenas unos años permitían la conexión a cualquier hora, sin restricciones, por una cuota mensual fija, hemos pasado a las conexiones de banda ancha y por cable, que proporcionan una capacidad cada vez mayor y que, pese al aumento de la demanda, comienzan a ir por delante del uso que se hace de ella. Este enorme aumento de velocidad, derivado de la fuerte competencia reinante en el mercado de las telecomunicaciones, propicia un caldo de cultivo ideal para la investigación y desarrollo de nuevas tecnologías que permitan una cantidad cada vez mayor de usos aplicables a la red de redes. Servicios de radio-televisión a la carta, videoconferencia o juegos en línea son algunos ejemplos de aplicaciones relativamente recientes que se han extendido rápidamente en Internet y que demandan conexiones domésticas de cada vez más capacidad. Esto propicia sin duda, aún más, el desarrollo de este tipo de conexiones y de forma cíclica la búsqueda de la próxima aplicación de éxito que rompa los esquemas preconcebidos de uso de Internet.

Si algo tienen en común todo este tipo de aplicaciones de nueva generación, y aquellas que están por venir, es el uso cada vez más intensivo de las conexiones de los usuarios para transferir grandes volúmenes de datos. Más allá, no sólo es necesaria capacidad de transferencia, sino garantizar una velocidad adecuada. Los usuarios no sólo quieren ver cine o televisión a través de Internet, quieren hacerlo sin cortes, sin saltos, de forma fluida, sin importar si el origen de los contenidos es la red o cualquier otro.

En este contexto, las aplicaciones de intercambio de información han evolucionado notablemente desde sus orígenes a la actualidad, adaptándose a los nuevos tiempos y requisitos que hacen necesarios un cambio de mentalidad a la hora de diseñarlas. Cada vez más gente tiene acceso a conexiones de alta velocidad y mayor el uso que hacen de ellas. Es por ello que las empresas, fundamentalmente aquellas cuyo negocio se basa en la distribución de contenidos digitales, necesitan cada vez más recursos para asumir la incesante y enorme demanda. Ante este mal uso, más y más compañías deciden enfocar sus esfuerzos no ya en ampliar la cantidad de recursos que ponen a disposición de sus clientes, lo cual resulta costoso y no garantiza resultados más que a corto plazo, sino en optimizar el uso que se hace de esos recursos, en cambiar la forma en la que la gente utiliza sus servicios, en proporcionar nueva vías de acceso a los contenidos que permitan el beneficio de la empresa y de sus clientes.

Las tecnologías tradicionales de distribución de información basadas en el modelo cliente-servidor son ineficientes, tal como si un sólo agricultor tuviese que dar de comer con su esfuerzo y trabajo de todo un año a todo un pueblo. Mientras la población de ese pueblo se mantiene limitada, el agricultor puede ser capaz de producir para todos los que lo necesiten, pero conforme la población aumente cada vez será más difícil. Podrá contratar jornaleros, pero llegará un momento en que ni todo el esfuerzo humano disponible será suficiente para alimentar a una gran ciudad, ya que el campo de cultivo no producirá más por mucho que se sumen personas que lo trabajen.

La alternativa a este modelo es la colaboración. Todo aquel que desee una parte de la cosecha debe ayudar en su recogida. De este modo, el beneficio de todos es el beneficio individual y viceversa. Este concepto tan intuitivo lleva relativamente pocos años aplicándose a las redes de telecomunicaciones, en lo que se ha dado en llamar «redes de pares». Pese a ello, mucho ha evolucionado desde la época dorada de *Napster*, el precursor de este concepto, y mucho han evolucionado a su vez las tecnologías que se basan en la colaboración.

Siguiendo el símil agrícola, existe un problema aún peor, el problema de los aprovechados, aquellos que solicitan colaborar con el resto en la labor del campo, viven en la finca del agricultor, comen en su mesa, se llevan una parte de la cosecha al terminar la temporada, pero no han llegado a trabajar ni un sólo día, dando así buena cuenta del esfuerzo de los demás. Lamentablemente, este problema es extrapolable a las «redes de pares» en Internet, y es tan grave que provoca que todos sus beneficios desaparezcan conforme aumenta el número de usuarios que se aprovechan de los demás. Se hace necesario por tanto buscar una solución a este problema, no sólo en forma de incentivos que animen a colaborar, sino también en forma de sanción para evitar tales comportamientos en el futuro.

¿Cómo es posible garantizar, en un grupo de desconocidos reunidos para colaborar en la obtención de un recurso que el trabajo común se repartirá equitativamente y cada miembro de la comunidad se llevará una parte proporcionalmente justa al esfuerzo dedicado para conseguirla? La respuesta está en nuestro comportamiento, nuestra forma de actuar cotidiana, pese a que quizás no resulte evidente. Constantemente nos relacionamos y colaboramos con otras personas con las que no tenemos experiencia propia, en las que, pese a ello, confiamos. Cada vez que no tenemos hechos previos en los que basarnos, acudimos a la experiencia que otros relatan, a la «reputación» que ha obtenido ese desconocido en base a su trabajo, su interacción, con otros individuos en los que sí tenemos confianza. Parece lógico pues aplicar este razonamiento al problema que nos ocupa. Si no sabemos de antemano si el comportamiento de un desconocido va a ser adecuado o no, nos fiaremos de las experiencias que hayan tenido otros con él, y en base a ello tomaremos nuestras propias decisiones, confiando o no dependiendo de su reputación.

¿Por qué no aplicar este concepto a la red de redes, a Internet? Si constantemente debemos tomar decisiones sin nada en lo que basarnos, ¿por qué no hacerlo en las experiencias de los demás? Al fin y al cabo, Internet es un medio en el que millones de desconocidos colaboran constantemente, y en el que otros tantos se aprovechan del esfuerzo de los demás. Muchos han decidido poner en práctica este concepto en sus negocios electrónicos. Empresas tan conocidas como *Amazon* o la casa de subastas *eBay* ponen a disposición de sus usuarios la capacidad de juzgar y valorar la calidad de sus contenidos y de las transacciones comerciales que en ellas se realizan, sirviendo esas opiniones como punto de partida para crear una reputación que pueda ayudar a otros a decidirse a realizar una compra o una puja. Y lo más prometedor: su apuesta ha sido todo un éxito y la forma en la que gestionan

la reputación les ha llevado a lo más alto de sus segmentos de mercado.

Trataremos, por tanto, de aplicar este concepto al intercambio colaborativo de información en Internet, abriendo nuevas vías que permitan un uso más eficiente de los recursos de los que disponemos hoy día y que den lugar al desarrollo de aplicaciones cada vez más innovadoras que puedan disfrutar todo el que así lo desee.

Capítulo 2

Estado del arte

Internet fue concebida originalmente como una red que permitiese el intercambio de información entre dos puntos cualesquiera conectados de la misma. En este contexto, nace un modelo sencillo y efectivo para transferir datos, conocido como *cliente-servidor*, en el cual cada uno de los participantes toma un rol permanente de acuerdo con su función en la transacción. Un nodo servidor pone a disposición de los clientes la información, haciéndola pública mediante cualquier protocolo bien conocido como HTTP o FTP, y son los clientes los que solicitan al servidor una transferencia de la misma. Esto implica que cada vez que alguien desee una copia de un conjunto concreto de datos deberá ponerse en contacto invariablemente con el nodo que los sirve, obteniéndolos del mismo.

Pese a quela arquitectura cliente-servidor es un buen modelo en el aspecto técnico y de hecho es la base en la que se asienta Internet tal y como lo conocemos, acusa graves problemas de escalabilidad y disponibilidad al situar un único punto de fallo en los servidores de la información. Un corte accidental en una línea de comunicaciones puede dejar incomunicado un servidor y por tanto la información que ofrece no disponible. Pero no sólo una avería del servicio puede dar al traste con él, sino que un exceso de clientes solicitando el mismo recurso pueden acabar con el servicio. Por desgracia, y pese a que los recursos computacionales, tanto en capacidad de cálculo como en ancho de banda, son cada día mayores, siguen siendo finitos, y un exceso de demanda sobre un determinado servicio puede acabar con el mismo por completo.

A día de hoy, Internet es una red mucho más compleja y descentralizada que en sus orígenes, distribuída a lo largo de millones de nodos repartidos por todo el mundo que aparecen y desaparecen en cualquier momento. Esto encaja con el paradigma de «Red

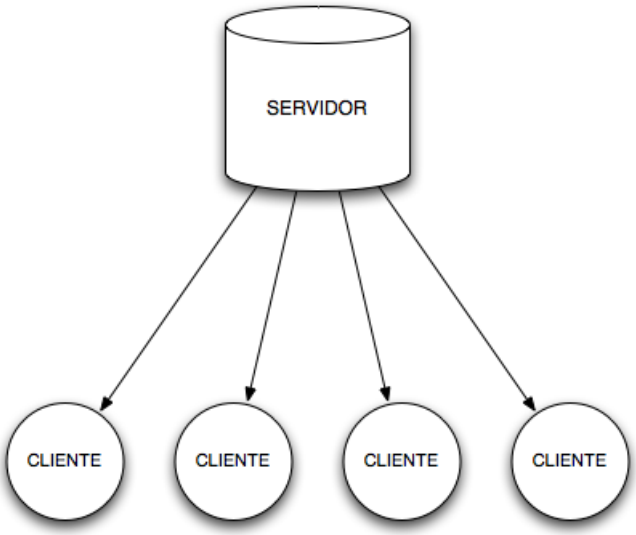

Figura 2.1: Modelo cliente-servidor

Transitoria Distribuída» o *DTN*, *Distributed Transient Network*, en el que se enmarcan las redes «ad-hoc» y las redes entre pares. También conocidas como *P2P*, suponen una alternativa ampliamente extendida a la tradicional arquitectura cliente-servidor. En contraposición a dicho modelo de comunicaciones, las redes de pares consisten en conjuntos dispersos de nodos que pueden actuar indistintamente como clientes o como servidores. Dichos nodos son denominados *pares*, y conforman una malla en la que cada uno de ellos colabora para conseguir un objetivo común, típicamente la obtención de un recurso que suele ser información en forma de datos. La figura 2.2 presenta un ejemplo de red de pares en la que algunos nodos (en verde) se limitan a compartir información con el resto y otros se benefician de los datos recibidos por los demás sin devolver nada a cambio (en rojo), mientras que el resto comparte los contenidos que recibe.

Es este carácter colaborativo de las redes de pares lo que las hace más atractivas y eficientes frente a otros modelos arquitectónicos, y de ahí su actual aceptación. Numerosas entidades y empresas han comenzado ya a implantar esta arquitectura para resolver cuellos de botella en sus servidores, especialmente cuando tienen que hacer frente a la entrega a sus usuarios de grandes volúmenes de información. El uso de estas redes permite, en el caso ideal, reducir la cantidad de veces que se debe transferir la información de las N (tantas como usuarios deseen obtener una copia) que imponen los modelos cliente-servidor a tan

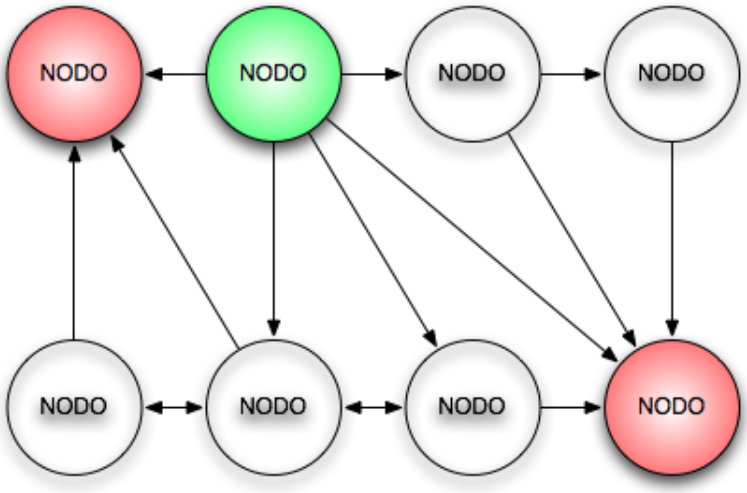

Figura 2.2: Modelo de red de pares.

sólo 1, necesaria para obtener una copia de la misma que se pueda compartir entre todos los interesados.

Algunas iniciativas surgidas recientemente como la «Participación Proactiva del Proveedor en Redes Punto a Punto», conocida popularmente como *P4P*[1], que ya se encuentran en estudio por parte de diversos proveedores, buscan la adecuación de las redes físicas actuales hacia el concepto de red de pares, de forma que las transacciones entre nodos ocurran preferentemente dentro de la red local al proveedor, optimizando así los recursos, aumentando la velocidad de transferencia y reduciendo los costes y la carga en los puntos de interconexión de red. Son precisamente este tipo de iniciativas y el ahorro de costes y recursos en general que suponen estas tecnologías lo que las hace tremendamente atractivas de cara a empresas y organizaciones que proporcionan servicios a través de Internet.

Si bien el intercambio de archivos es con diferencia el caso de uso estrella de este tipo de redes, no es ni mucho menos el único. El diseño versátil y flexible de las redes de pares permite su utilización en multitud de ámbitos, desde la telefonía y la televisión a través de Internet, como *Skype*[skype-p2p] o *Joost*[joost] respectivamente, a la computación distribuída en diversos campos como la bioingeniería, con proyectos como el «Proyecto de Investigación contra el Cáncer» de la universidad de Oxford y la Fundación National de

[1] *P4P* es una iniciativa de la «Asociación de la Industria de Computación Distribuída», *DCIA*, a la que ya se han sumado numerosas entidades y proveedores de servicios de Internet, como Verizon o el Grupo Telefónica en España.

Investigación contra el Cáncer. De esta forma es posible aprovechar sus ventajas en casi cualquier ámbito de las telecomunicaciones modernas. Dichas ventajas son:

Escalabilidad El principal punto a favor de las redes de pares, que permite aumentar potencialmente hasta el infinito el número de clientes de un servicio sin que éste se resienta. Es precisamente el carácter colaborativo de estas redes lo que hace posible su mayor escala frente a otras arquitecturas.

Disponibilidad O lo que es lo mismo, la capacidad para mantener funcionando un servicio aún cuando la fuente original del mismo deje de funcionar o incluso desaparezca. En el caso de las transferencias de información el ejemplo es inmediato, ya que pese a que alguien tiene que introducir inicialmente los datos en la red, conforme más usuarios obtengan copias de los mismos, mayor será su disponibilidad. De este modo es indiferente un fallo puntual en un nodo que contenga los datos, ya que muchos otros podrán servirlos.

Descentralización La ausencia de puntos que ofrezcan servicios fijos. Si bien pocas redes de pares se pueden considerar puras en este aspecto, ya que la mayoría de las actuales requieren al menos el uso de servidores en los que agrupar a todos los usuarios de la red, en general se cumple el objetivo de no mantener servidores centralizados cuyo funcionamiento sea imprescindible para la red.

Adicionalmente se suelen requerir otro tipo de características a las redes de pares que por desgracia a día de hoy no están muy extendidas en las redes más populares.

Anonimato La capacidad para mantener anónimos a los usuarios de la red, de forma que nadie (ni siquiera otros usuarios de la red) puedan obtener información personal de ningún tipo.

Seguridad Mantener autenticados (y, opcionalmente, autorizados) a los usuarios de la red, así como garantizar el origen de los contenidos que se distribuyen, que no han sido alterados en el proceso de transferencia, o incluso que permanecen confidenciales en caso de que así se necesitase.

Equidad El reparto de forma justa de los recursos disponibles por cada uno de los nodos que conforman la red en beneficio común. Todos los participantes de una red de pares deben colaborar en la medida de sus posibilidades para que el beneficio se maximice.

A pesar de que en la actualidad existen redes de pares que implementan algunas de estas características, su uso aún no se ha generalizado. El problema no es tanto su implementación, sino asegurarse de que no tengan contrapartidas graves y de que obtener alguna de estas cualidades deseables no implique perder otras necesarias.

Un ejemplo de ello es la red *Freenet*[freenet], un sistema de publicación punto a punto anónimo, en el que los archivos son almacenados en un conjunto de servidores que se prestan de forma voluntaria. Dicho conjunto es dinámico -los servidores se pueden unir y dejar la red en cualquier momento. Los archivos publicados se copian a un subconjunto de servidores, y cada vez que se reenvían a uno nuevo la dirección de origen asociada a los mismos puede cambiarse por un valor aleatorio. De esta forma, dicha información es básicamente inútil a la hora de determinar el origen de un archivo, por lo que éstos pueden publicarse de forma totalmente anónima.

Las peticiones de archivos siguen la misma filosofía, al ser reenviadas de servidor en servidor y almacenar (opcionalmente) cada uno de los servidores intermedios una copia en su caché. Así, *Freenet* es además resistente a la censura y evita que los registros de actividad de cada servidor sirvan para trazar el destino final de un archivo.

Sin embargo, si bien mediante esta arquitectura conseguimos el anonimato de los usuarios, la contrapartida es que no tenemos tampoco ninguna certeza sobre la veracidad de los contenidos ni capacidad para identificar a aquellos que hacen un mal uso de la red. El problema es, por tanto, no ya la dificultad de implementar las características propuestas, sino incluso la imposibilidad de proporcionar algunas de ellas en conjunción.

El mayor problema cuando hablamos de redes de pares aparece cuando no se cumple el principio de equidad en el reparto de los recursos, ya que cuanta menos colaboración exista por parte de los integrantes de la red de pares, más se asemeja a una arquitectura clásica de cliente-servidor y, por tanto, dejan de apreciarse las ventajas comentadas frente a ésta.

El máximo exponente de este problema es el *free-riding*, considerado de forma general como el abuso por parte de aquellos que consumen más recursos de los que retribuyen al sistema. El concepto es sencillo y se basa en las reflexiones realizadas por Garret Hardin

en su artículo *The Tragedy of the Commons[commons]* publicado en la revista *Science* en 1968. El abuso por parte de todos aquellos que componen una comunidad en busca exclusivamente del beneficio propio acaba derivando en el perjuicio de la comunidad al completo y por ende de todos sus integrantes.

Las redes de pares, al basar su efectividad precisamente en la colaboración de aquellos que las integran, son extremadamente vulnerables a este problema. Cuando se habla de *free-riding* en el caso particular de estas redes, el concepto se especifica ligeramente más allá en comparación con la acepción general, muy aceptada en ámbitos económicos y el estudio de la teoría de juegos. Por ello, se entiende como *free-rider* en una red de pares a aquel nodo de la misma que obtiene un recurso (por ejemplo un fichero) sin devolver nada a cambio al resto de nodos que puedan estar interesados en ese mismo recurso. Este tipo de comportamientos, cuando se generalizan, derivan en que la relación entre clientes y servidores que presentan las redes de pares como principal ventaja frente a la arquitectura cliente-servidor se reduzca a una relación de uno a uno -es decir, los recursos se comparten entre servidor y cliente, sin que medie colaboración entre clientes de por medio-, suponiendo entonces la degradación de la red hasta un nivel tal en el que no aporta ninguna ventaja frente a otros esquemas. En la figura 2.3 se presenta el problema en su máxima expresión, en caso de que sólo el nodo que posee la copia original del documento la comparte con los demás, que se limitan a descargar de él. Como se puede apreciar, la arquitectura de la red es en esta situación idéntica a la del modelo cliente-servidor.

Si bien la teoría de juegos se ha ocupado de este problema en múltiples ocasiones, llegando a la clara conclusión de que, a corto plazo, para el individuo es más ventajoso obtener recursos aprovechándose de los esfuerzos de los demás, en la práctica apenas se ha investigado una forma de solventar el problema en las redes de pares; o al menos, encontrar una manera en la que un comportamiento cooperativo ofrezca más ventajas que uno puramente egoísta. Así las cosas, pocas son las redes de pares existentes en la actualidad que ofrecen mecanismos para combatir el *free-riding*, y lo que es aún peor, aquellas que implementan algún tipo de medida, no consiguen eliminar el problema; ni tan siquiera reducirlo.

Estos mecanismos son realmente básicos y se centran más en incentivar un comportamiento correcto en los usuarios de la red en lugar de evitar un comportamiento egoísta. Algunas implementaciones como la de la red *eDonkey*[edonkey-spec] se basan en sistemas

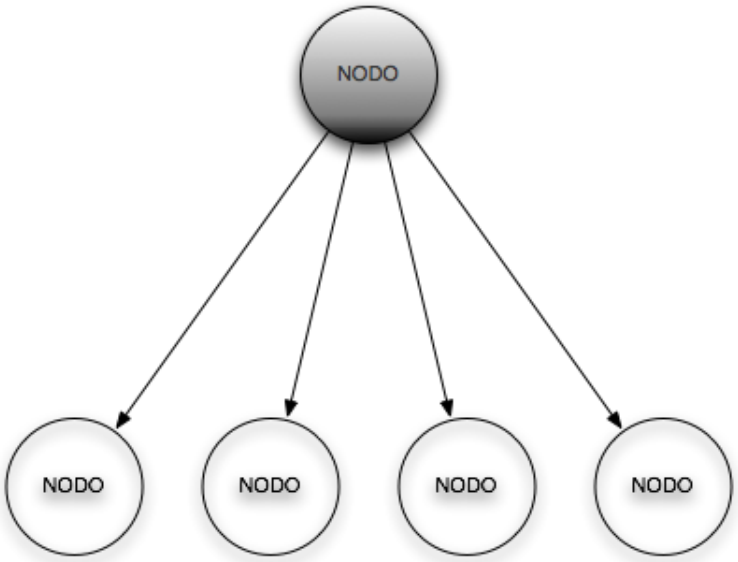

Figura 2.3: Red de pares afectada por el problema del *free-riding*.

de créditos que permiten a los usuarios avanzar puestos en las colas de descarga y por tanto agilizar la obtención de contenidos. Otras redes como *BitTorrent*[bt-spec] incorporan mecanismos más drásticos como el bloqueo de aquellos usuarios de los cuales se observa un comportamiento inadecuado (conocido como *choking* o «estrangulamiento»), pero fallan en su objetivo al desbloquear de forma optimista y periódica a otros nodos para solucionar el problema del arranque inicial o *bootstrapping*.

Por tanto, no existen propuestas concretas y efectivas que permitan abordar de forma eficaz el problema del *free-riding* en las redes de pares más populares a día de hoy. Dado que cuanto más se extiende su uso, más se agudiza el problema, se antoja necesario buscar una solución que pueda llevarse a la práctica e implantarse de forma sencilla.

2.1. *Free-riding* en las redes de pares

Para proponer soluciones a un problema, es necesario primero comprenderlo en toda su extensión, y para ello debemos analizarlo en la mayor cantidad posible de ámbitos y entender por qué se produce en cada uno de los casos sometidos a análisis. Comenzaremos por tanto presentando un estudio de algunas de las redes de pares más populares y extendidas en la actualidad y del impacto que tiene el problema del *free-riding* sobre ellas.

Red	¿Permite búsquedas?	¿Es vulnerable?	¿Implementa medidas?
Freenet	Sí	Sí	No
Gnutella	Sí	Sí	No
eDonkey 2000	Sí	Sí	Sí, sistema de créditos
BitTorrent	No	Sí	Sí, bloqueo eventual

Cuadro 2.1: Algunas características significativas de las redes analizadas.

2.1.1. Freenet

Freenet es una red de pares descentralizada y diseñada con el ánimo de facilitar el anonimato y eludir la censura, especialmente pensada para aquellas redes de telecomunicaciones que estén sometidas a la monitorización y control. Su objetivo es el almacenamiento y consulta de documentos mediante una clave asociada a los mismos que permite su protección. Conforma una red no jerarquizada de nodos que comparten mensajes y documentos entre ellos.

Las peticiones de *Freenet* se reenvían de un nodo al siguiente acorde con decisiones locales sobre qué receptor potencial puede progresar en mayor medida hacia el archivo que se solicita. De esta forma, cada petición va encaminada no a un nodo concreto, sino a cualquiera que pueda ofrecer una copia del documento deseado, ya que los propios nodos tienen la habilidad de almacenar en una memoria caché los contenidos a los que tienen acceso.

Para recuperar un documento publicado en *Freenet* es necesaria la clave que describe dicho documento, y que será utilizada en sucesivas peticiones. Mediante dicha clave, cada nodo puede decidir cuales de sus nodos vecinos puede ayudarle a recuperar la información, reenviándole por tanto la petición. De este modo, es posible llegar bien al nodo que originalmente contiene el documento, bien a alguno intermedio que ha obtenido previamente una copia y la ha almacenado en su propia caché, utilizando un sencillo algoritmo de búsqueda en profundidad. En la figura 2.4 se presenta un ejemplo de petición en la red *Freenet*, de forma muy simplificada, que se corresponde con la clave 10, 32, 1, 34, 19, 36, 46.

En una red como la descrita, un *free-rider* es aquel que, si bien obtiene documentos del resto de nodos, no reenvía las peticiones que le llegan ni las contesta por su cuenta en caso de que él mismo tenga una copia del contenido solicitado. *Freenet* lidia con esta forma de *free-riding* simplemente ignorando este tipo de nodos. Cada nodo mantiene actualizada una

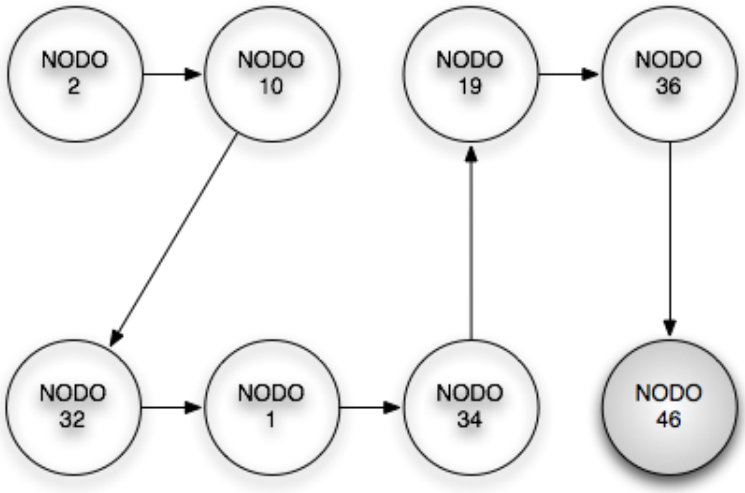

Figura 2.4: Ejemplo de petición en la red *Freenet*.

tabla de rutas que describe a sus vecinos, de forma que al observar un nodo que no contesta a sus peticiones lo marcará como inusable y tratará de evitar enviarle sucesivas peticiones. Es por ello que *Freenet* no proporciona ningún mecanismo real para evitar el *free-riding*, sino que simplemente lo ignora para paliar sus efectos, siendo totalmente ineficiente a la hora de disuadir este tipo de comportamientos. Esto provoca una sobrecarga debida a las peticiones de los *free-riders* y no deja ningún tipo de beneficio en el resto del sistema.

2.1.2. Gnutella

La red *Gnutella* es una alternativa descentralizada a otras redes de pares semi-centralizadas como *Napster* o *eDonkey 2000*, que adquirió gran repercusión precisamente tras el cierre de la primera de éstas.

En *Gnutella*, cada nodo o *par* intenta mantener un número pequeño (alrededor de tres) de conexiones simultáneas activas con otros pares. Éstos se seleccionan de una lista con las direcciones de los pares que conoce el nodo. Otros pares pueden descubrirse mediante una amplia variedad de mecanismos, como por ejemplo monitorizar mensajes de `PING` y `PONG`[2], apuntar las direcciones de pares que inician consultas, recibir peticiones de pares desconocidos previamente o usar canales fuera de banda como por ejemplo páginas web.

[2]Los mensajes de `PING` y `PONG` son mensajes definidos por el protocolo *Gnutella*[gnutella-spec] para el descubrimiento activo de nuevos pares en la red.

Aún así, no todos los pares descubiertos de alguna de estas maneras aceptarán nuevas conexiones, ya que pueden haber alcanzado su propio límite de conexiones o incluso desconfiar de otros pares que no conozcan. Este proceso es por tanto complejo y requiere tiempo, y se ve agravado por aquellos pares que dejan la red y que provocan que aquellos que aún permanecen en la misma intenten reemplazar las conexiones perdidas.

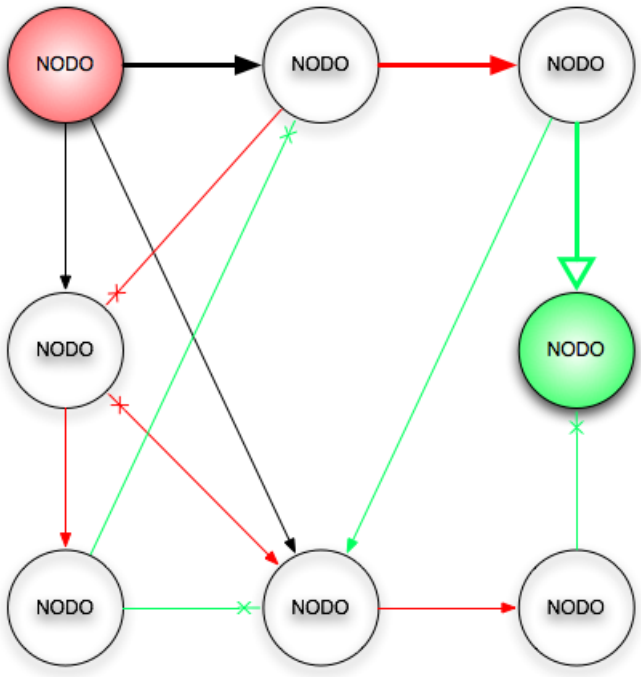

Figura 2.5: Ejemplo de petición en la red *Gnutella*.

Las peticiones de *Gnutella* se propagan por la red por el método de inundación o *flooding*. Es decir, cada par o nodo reenviará las peticiones recibidas a todos los nodos a los que está conectado, que a su vez repetirán en cadena la misma forma de actuar. Si un par es capaz de satisfacer una petición, responderá al nodo que la originó y, en cualquier caso, también la reenviará siguiendo un algoritmo de búsqueda en amplitud que permitirá encontrar todos los nodos posibles que puedan responder a la misma. La figura 2.5 presenta un ejemplo de petición que parte del nodo original (en rojo) y se extiende por cada uno de los nodos a los que se encuentra éste conectado. Este algoritmo se repite iterativamente de forma que cada nodo reenvía la petición a aquellos de sus vecinos de los cuales no ha recibido aún la petición, lo cual deriva en la duplicación y rechazo de algunos

de esos reenvíos, hasta llegar al nodo que sirve los contenidos solicitados (en verde).

La red *Gnutella* no está exenta del problema del *free-riding*, tal y como se demuestra en el artículo publicado por dos investigadores de Xerox[free-riding-gnutella]. *Gnutella* es vulnerable debido a que sus pares no mantienen ningún tipo de información de estado sobre otros pares, lo cual imposibilita discernir entre aquellos que tienen un comportamiento adecuado y aquellos que no (*free-riders*). En particular, éstos seguirán recibiendo peticiones incluso si nunca responden a ninguna de ellas. De este modo, la presencia de este tipo de nodos que se aprovechan de la red provocarán que ésta se disperse y que las peticiones tengan que seguir caminos más largos cada vez para encontrar la información solicitada.

Como se puede inferir, el problema del *free-riding* en la red de pares *Gnutella* no sólo degrada el rendimiento de la misma, sino que puede llegar a provocar que las búsquedas o peticiones de archivos existentes lleguen a fallar si los nodos que abusan de la misma consiguen segmentarla o hacen que los caminos necesarios para alcanzar la información superen un umbral prefijado de saltos entre nodos.

2.1.3. eDonkey 2000

eDonkey es una red de intercambio de archivos originada por el programa del mismo nombre. Se trata de una red de pares semi-centralizada, en la que los pares se conectan a servidores que sirven de punto de confluencia para realizar búsquedas y poner nodos en contacto entre sí. Dichos servidores albergan los metadatos con toda la información de los archivos compartidos por sus actuales usuarios, de forma que es posible realizar búsquedas de contenidos aún cuando sus propietarios se encuentran desconectados. De esta forma, el modelo inicial de la red es parecido a una arquitectura cliente-servidor en la que los clientes colaboran para la obtención de un objetivo común, pero siguen siendo altamente dependientes de puntos centrales de la red.

Numerosos clientes han implementado a lo largo del tiempo el protocolo original *eDonkey 2000*, ampliándolo y mejorándolo, como por ejemplo *eMule*, el más famoso de todos ellos. Actualmente la red es más independiente de los servidores y permite hacer búsquedas a lo largo y ancho de la misma, sin necesidad de restringirlas únicamente al servidor al que nos encontramos conectados en el momento de realizar la búsqueda.

La red sobre la que se asienta *eMule* va más allá de la original y se denomina *Kad*,

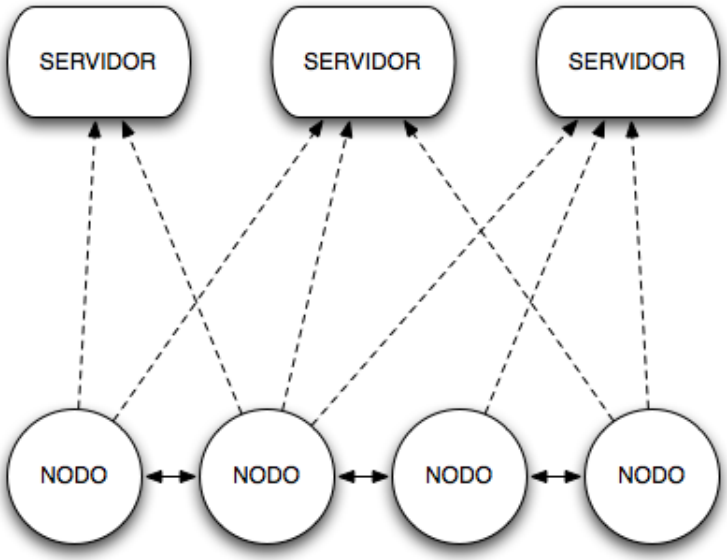

Figura 2.6: Esquema de la red *eDonkey* original.

siendo una implementación concreta del protocolo *Kademlia*[kademlia][3] en la que se basaba la red *Overnet* original[4]. El conjunto de mejoras aplicadas sobre la red original permite no sólo obtener nuevas funcionalidades, sino implementar medidas para potenciar el buen uso de la red. Lamentablemente, dichas medidas no evitan el problema del *free-riding*. La actual red *Kad* proporciona un sistema de incentivos o créditos que permiten, a grandes rasgos, acelerar los tiempos de descarga. Los créditos se obtienen compartiendo contenidos directamente con otros pares, y son almacenados para su posterior uso. Una vez se desee realizar una descarga de otro par conectado a la red, se podrán presentar créditos en caso de haberlos obtenido previamente de ese par (y no de ningún otro) para subir posiciones en las colas de espera para descarga. Dado que dichas colas suelen ser numerosas, el sistema de créditos supone un claro aliciente, pero no evita en absoluto que los pares conectados a la red descarguen contenidos sin haber compartido nada previamente. Más en concreto,

[3]Kademlia es un protocolo de la capa de aplicación diseñado para redes de pares descentralizadas, que permite el intercambio de información entre los nodos así como la regulación de dicho intercambio, todo ello utilizando el protocolo no orientado a conexión *UDP*. *Kademlia* no requiere servidores, por lo que soluciona el problema de la entrada al sistema o *bootstrapping* asumiendo que el cliente que entra por primera vez a la red conoce al menos un nodo que soporta el protocolo. Su funcionamiento es muy similar al de la red *Gnutella*, dispersando las consultas y búsquedas por inundación a través de los nodos que componen la red, lo cual garantiza resultados positivos en un escenario ideal.

[4]Surgida a partir de la red nativa sin servidores del ya desaparecido programa *eDonkey 2000*, *Overnet* implementa una variante del protocolo *Kademlia*.

cuando las colas de descarga se vacíen, los nodos sin créditos podrán descargar de otros pares, por lo que dichos créditos no son en absoluto necesarios para utilizar la red.

Es por esto que tanto la red *eDonkey* original, así como sus sucesoras, *Overnet* y *Kad*, son vulnerables al problema del free-riding, al no imponer ningún sistema obligatorio para obtener beneficios de otros nodos de la red. Si bien existe una base sobre la que intentar solventar el problema, no es sencillo, ya que la idea intuitiva de obligar a obtener créditos para poder descargar algún contenido dificulta la incorporación de nuevos nodos en el sistema. En caso de que la obtención de créditos fuese necesaria para utilizar el sistema, los pares se verían obligados a proporcionar contenidos por su cuenta, obtenidos fuera del sistema, para poder empezar a utilizarlo, sin mencionar tan siquiera los problemas que podrían tener aquellos clientes tras un *NAT*[nat-rfc]. En la implementación actual de la red prima la sencillez de uso y de inicio en el sistema por encima de cualquier otro requisito, lo cual ha derivado en su rápida y extendida aceptación, pero tiene como contrapartida los elevados tiempos de descarga necesarios en líneas generales para obtener un archivo.

2.1.4. BitTorrent

BitTorrent es una red de pares diseñada por Bram Cohen y que tiene numerosas similitudes con otros protocolos como *eDonkey 2000*. Cohen es autor no sólo del diseño y especificación del protocolo[bt-spec], sino también de la primera implementación del mismo en el lenguaje *Python*. La peculiaridad de *BitTorrent* es el uso de archivos «torrent» que sirven tanto para localizar servidores que puedan poner en contacto a todos los usuarios interesados en un archivo, llamados *trackers*[5], como para asegurar la integridad de los mismos, ya que estos archivos incluyen adicionalmente resúmenes *SHA1* de cada una de las porciones o «pedazos» en las que se divide un archivo que forma parte del «torrent».

Dado que el protocolo no proporciona ninguna facilidad de búsqueda per se, los usuarios de esta red deben obtener los archivos «torrent» por otras fuentes, típicamente páginas web especializadas en el almacenaje y distribución de este tipo de archivos. Una vez conseguido uno de estos archivos, el cliente se conectará al *tracker* indicado en el propio «torrent» para obtener un listado de otros usuarios que tengan una copia de los contenidos que

[5]Literalmente, «rastreador» en inglés. El término designa la idea de que estos servidores llevan el rastro de los usuarios que descargan un determinado «torrent» y por tanto tienen un mapa completo de los usuarios interesados en unos contenidos concretos.

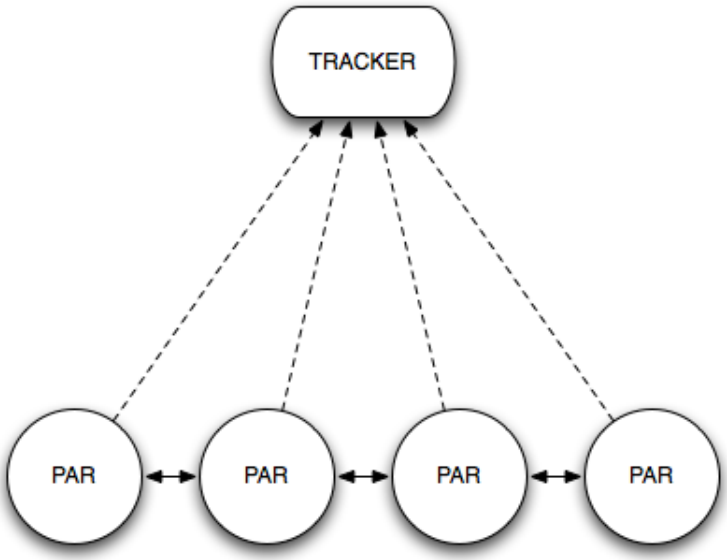

Figura 2.7: Esquema de la red *BitTorrent* asociada a la descarga de un «torrent».

se desea descargar, o que están también en proceso de descarga. Con esta información, el cliente se encuentra en disposición de contactar directamente con otros nodos de la red para solicitarles colaboración en la descarga. Los clientes refrescan periódicamente la información obtenida de los *trackers* para garantizar que la mayor cantidad posible de nodos referenciados siguen conectados a la red.

Adicionalmente, existen extensiones oficiales al protocolo original que permiten descentralizarlo por completo mediante el uso de Tablas Hash Distribuídas o *Distributed Hash Tables* (*DHT*). A grandes rasgos, y mediante el uso del protocolo *Kademlia* ya mencionado previamente, cada par de la red se convierte en un «nodo»[6] que hace a su vez las funciones de *tracker* y cliente del protocolo. De este modo es posible deslocalizar los «torrent» y hacer que no dependan de ningún *tracker* concreto, derivando en una red de pares pura en la que no hay puntos centrales de fallo ni cuellos de botella. En la figura 2.8 se ilustra el esquema de una red *BitTorrent* que implementa el protocolo *DHT*. En ella, todos los pares de la red se pueden conectar entre sí y algunos de ellos, denominados «nodos», actúan además como *trackers* de la red.

[6]Si bien a lo largo de este documento se hace uso casi indistinto de los términos «par» y «nodo», en el caso concreto del protocolo *DHT* se utiliza una convención por la cual se denomina «par» a los clientes que implementan el protocolo *BitTorrent* original, y «nodo» a aquellos que adicionalmente implementan el protocolo *DHT*[dht-spec].

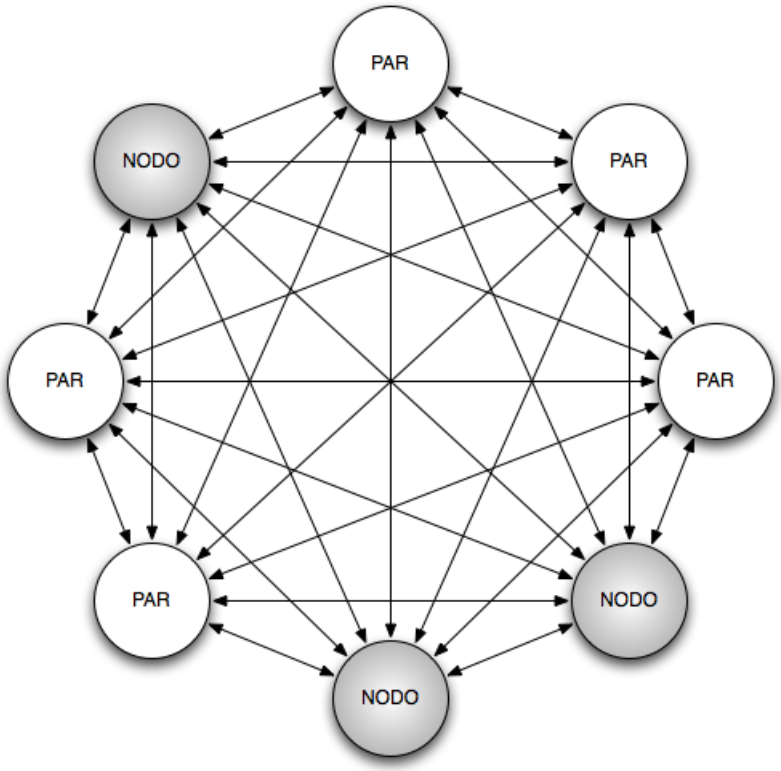

Figura 2.8: Esquema de una red *BitTorrent* con el protocolo *DHT*.

El protocolo incorpora desde sus inicios mecanismos para penalizar a aquellos pares de la red que se niegan a compartir contenidos con otros. Cuando, tras una petición a un par, este no devuelve una respuesta satisfactoria, el cliente de *BitTorrent* marcará a dicho par en un estado de bloqueo o *choking*, lo que evitará que el cliente comparta a su vez contenidos con dicho par. De este modo se penaliza a los *free-riders* de la red, si bien la medida es insuficiente. Debido al problema del inicio en el sistema o *bootstrapping*, el protocolo define mecanismos que permiten a los recién llegados obtener datos que pueden compartir con otros pares y de este modo comenzar a relacionarse en la red. Es este mecanismo conocido como desbloqueo «optimista» u *optimistic unchoke* precisamente lo que hace que *BitTorrent* sea vulnerable al problema del *free-riding*. Los clientes del protocolo, de forma periódica, desbloquean otros pares bloqueados con la esperanza de que con los datos que se compartan con ellos puedan empezar a compartir a su vez con otros pares y así iniciarse en el sistema. De esta forma, un *free-rider* no tiene más que hacerse pasar por un recién llegado para conseguir que otros pares de la red compartan bloques con él y así poder

descargar archivos en su totalidad sin compartir nada en retribución.

Lo que es peor, existen demostraciones prácticas de que el diseño original del protocolo no sólo no evita el *free-riding*, pese a tenerlo en cuenta desde sus inicios, sino que incluso lo potencia. *BitThief* [bt-freeriding] es un cliente de *BitTorrent* especialmente programado para tratar de obtener contenidos de dichas redes sin compartir datos con otros nodos. Experimentos realizados con *BitThief* demuestran que, en igualdad de condiciones respecto a otro clientes como el oficial, éste es capaz de descargar los mismos contenidos de forma significativamente más veloz. Más allá, penalizando el cliente *BitThief* y permitiéndole descargar solamente de otros nodos que estén a su vez en proceso de descarga, es decir, no tengan una copia completa de los archivos deseados, frente al cliente oficial que no tiene ningún tipo de limitación, gracias al *free-riding* es capaz todavía de descargar los contenidos en el total de los casos, si bien los tiempos obtenidos son mucho mayores en comparación con el cliente *BitTorrent* oficial.

Lamentablemente, estos experimentos demuestran que pese a que el protocolo tiene en cuenta en su diseño el problema de los nodos que se aprovechan de los recursos de la red o *free-riders*, no es capaz de evitarlo de forma satisfactoria. Es por ello que para el presente proyecto elegimos el protocolo *BitTorrent* como punto de partida para las investigaciones que permitan abordar el problema del *free-riding*, ya que es a día de hoy el único que ya incorpora medidas conscientes para combatirlo, sin olvidarnos de la creciente popularidad que está ganando esta red de pares y del excelente resultado que proporciona en un escenario ideal, siendo una de sus mayores características la alta velocidad a la que se pueden llegar a descargar los contenidos publicados a través de ella.

Dada la necesidad para nuestros propósitos de extender el protocolo *BitTorrent* original, hay un aspecto que no se debe descuidar del mismo, y es la incorporación de mecanismos de extensión similares a los de otros protocolos como *TCP*. El mensaje inicial o *handshake* en el protocolo *BitTorrent* incorpora un total de ocho *bytes* reservados para extensiones, lo que permite poner de acuerdo a dos clientes en las extensiones que implementan y van a utilizar tras la negociación. Esto proporciona un total de sesenta y cuatro extensiones posibles, correspondiéndose cada una a uno de los *bits* reservados a tal efecto. De este modo el protocolo se encuentra muy limitado, y a lo largo del tiempo se ha hecho patente la necesidad de mejorar este mecanismo de extensión. A día de hoy se han desarrollado hasta tres extensiones conocidas que amplían el protocolo original hasta un número indefinido

de nuevas extensiones. Cada una de estas tres extensiones, el *Protocolo de Mensajería de Azureus*[azmp-spec], el *Protocolo de Extensión de LibTorrent*[ltep-spec] y el *Protocolo de Extensión de BitComet*[bcep-spec] proporcionan funcionalidades similares en el ámbito de sus propios clientes de *BitTorrent*. Las más populares a día de hoy son las implementadas por *Azureus* y *LibTorrent*, y si bien no son incompatibles, si es necesario un protocolo sencillo que permita seleccionar una de entre las dos en la negociación inicial entre dos pares. Afortunadamente existen propuestas concretas como el *Protocolo de Negociación de Extensiones*[enp-spec] que han sido implementadas por los principales clientes que soportan ambas extensiones. Es por ello que, a falta de algún convenio oficial, se hace necesario adoptar un estándar de facto que en la actualidad rige que cualquier extensión no oficial al protocolo *BitTorrent* original debe realizarse mediante alguno de los protocolos de extensión existentes.

Capítulo 3

Objetivos

El objetivo de este proyecto es tratar de solventar el problema, común a las redes de pares, de los nodos que abusan de las mismas obteniendo recursos de ellas sin devolver nada a cambio, conocido como *free-riding*. Centraremos nuestro estudio en la red *BitTorrent*, de gran auge en la actualidad, consistente en un protocolo extenso y lleno de detalles que quedan al margen de nuestro interés, por lo que partiremos de una implementación existente que será modificada para incluir la solución aquí propuesta. En este caso se ha elegido *Vuze*[1], por ser el cliente de uso mayoritario a día de hoy y por implementar numerosas ampliaciones sobre el protocolo, así como por tratarse de *software libre*, de forma que es posible acceder a su código fuente, modificarlo y extenderlo, y por estar escrito en el lenguaje *Java*, lo cual facilita la utilización de otras librerías necesarias para la consecución de nuestros objetivos, como veremos más adelante.

Así mismo, y dado el estado actual de los mecanismos de extensión del protocolo expuestos en el capítulo anterior, y dado que la implementación se basará en el cliente *Vuze*, se hará uso de su propio «Protocolo de Mensajería de Azureus» para extender el protocolo original con los mensajes e interacciones descritas por la solución propuesta.

Por tanto, nos marcaremos una serie de hitos que trataremos de cumplir de forma consecutiva para llegar al objetivo final ya planteado, la resolución del problema del *free-*

[1]Cliente del protocolo *BitTorrent* muy conocido por su antiguo nombre *Azureus*, ampliamente extendido y que incorpora numerosas y significativas mejoras sobre el protocolo original, tales como las Tablas Hash Distribuídas (*Distributed Hash Tables*) o la habilidad para descentralizar por completo el protocolo con respecto a sus *trackers* mediante la inclusión de más de un punto de distribución en los «torrents» de forma que en caso de fallo de uno de ellos, se pueda acudir a otro para obtener la relación de nodos en proceso de descarga. El cliente se puede descargar, junto con un amplio conjunto de información, de su propia página web[vuze].

riding en el protocolo *BitTorrent*:

1. Estudiar en detalle el protocolo *BitTorrent* y analizar las causas de su vulnerabilidad ante el *free-riding*.

2. Basándonos en lo anterior, analizar las medidas necesarias para contrarrestar esas causas y, por tanto, el problema que nos ocupa.

3. Diseñar una arquitectura completa que permita llevar a la práctica dichas medidas sobre el protocolo original de *BitTorrent*

4. Una vez completado y refinado el diseño, obtener una implementación del mismo a partir del cliente de *BitTorrent Vuze*.

Capítulo 4

Diseño e implementación

A continuación y a lo largo de todo este capítulo se presenta una descripción completa del conjunto de protocolos, mensajes y procedimientos que conforman la arquitectura propuesta como solución al problema del *free-riding* en *BitTorrent*. Se ahondará no sólo en la descripción de la misma, sino en la implementación realizada sobre el cliente y tracker de *BitTorrent* proporcionados por el *software Vuze*, anteriormente conocido como *Azureus*.

4.1. Definiciones

Antes de entrar en detalles puramente técnicos sobre la arquitectura, es necesario refrescar algunos conceptos así como definir otros que se utilizarán en adelante y que es necesario tener claros para la compresión de este capítulo.

tracker o «rastreador». Servidores del protocolo original *BitTorrent* que gestionan listas de usuarios interesados en un determinado «torrent». En el caso que nos ocupa, además, gestionarán la identidad y reputación de los usuarios de la red de los cuales sean responsables.

par del inglés *peer*. Cada uno de los nodos iguales que conforman una red *BitTorrent*.

cliente el *software* que implementa el protocolo *BitTorrent* utilizado por el usuario. En adelante se utilizará este término para referirse al usuario del protocolo en el que se sitúa en cada momento el punto de vista. El resto de nodos de la red que interactúen con el cliente serán denominados, sencillamente, «pares».

home tracker el *tracker* del protocolo *BitTorrent* que además se encarga de la gestión de la identidad y reputación del cliente (véase la definición de cliente).

remote tracker el *tracker* del protocolo *BitTorrent* al que se encuentra conectado el cliente en el instante actual. Si bien debe implementar las extensiones aquí descritas al protocolo original, dicho *tracker* no es responsable ni del cliente ni del par participantes en una transacción. Nótese que en una red *BitTorrent* que implemente la extensión *DHT*, el *remote tracker* coincidirá con el primer «nodo» de la lista incluída en el «torrent» con el que el cliente pueda establecer una conexión.

home tracker del par el *tracker* del protocolo *BitTorrent* que además se encarga de la gestión de la identidad y reputación del par objeto de una transacción.

seeder o «semilla», aquel nodo o par de la red que tiene una copia completa de un determinado contenido y la comparte desinteresadamente sin recibir nada a cambio.

leecher o «sanguijuela», el resto de nodos o pares de la red que no son *seeders*, es decir, aquellos que se encuentran en proceso de descarga de un contenido y que aún no han obtenido una copia completa[1].

transacción intercambio de información productiva entre dos nodos de la red, generalmente, un bloque o «pedazo» completo de un archivo en cuya descarga colaboran.

4.2. Solución propuesta

Como ya se ha expuesto, las medidas implementadas en *BitTorrent* para evitar el *free-riding* son insuficientes debido a que, para permitir a nuevos nodos comenzar a descargar un nuevo archivo, comparten datos con los mismos de forma arbitraria. Esto implica que un *free-rider* puede hacerse pasar por un recién llegado para obtener beneficios sin compartir nada. Para evitar esto, por tanto, es necesario identificar a los nodos de la red de forma que no puedan hacerse pasar por nodos nuevos.

De forma estándar, un «torrent» está asociado a un *tracker* y por tanto los clientes que deseen descargar ese contenido deberán conectarse a dicho *tracker*. Por eso el esquema

[1] Comúnmente, se denomina erróneamente *leecher* al *free-rider*, es decir, aquel que descarga contenidos sin compartir nada a cambio con el resto de nodos de la red.

implementado necesariamente debe permitir a los clientes la capacidad de identificarse y ser autenticados independientemente del *tracker* al que se hayan conectado y del contenido que estén descargando. De este modo se convierte en un requisito utilizar técnicas de identidad digital con todos y cada uno de los pares de la red, que permitan mantener un seguimiento permanente, unívoco y ubicuo de cada uno de ellos. Así se consigue, por tanto, que los usuarios de la red estén identificados en todo momento y adicionalmente que conserven dicha identidad independientemente del *tracker* al que se conecten.

Una vez sea posible identificar a los usuarios de la red de pares, será posible almacenar cualquier tipo de información acerca de ellos. En el caso que nos ocupa, la información deseable es aquella que describa el comportamiento del nodo a lo largo del tiempo en la red, no sólo cada vez que se conecte a un determinado *tracker*, sino cada vez que haga uso del protocolo. De este modo, cada vez que un cliente de *BitTorrent* solicite una transacción, su par podrá obtener información acerca del comportamiento del cliente a lo largo del tiempo y en base a ella, tomar una decisión acerca de la conveniencia o no de compartir datos con él. Evidentemente, no sólo debemos poner un grado de confianza en la identidad de los nodos de la red, sino también en la reputación -esto es, la consideración o estima- que observemos para cada uno de ellos, lo cual implica de nuevo la necesidad no sólo de la identidad digital, sino además de servicios de firma digital y validación de las mismas, que garanticen la autenticidad de las reputaciones y a su vez la propia fiabilidad de aquellos que las notifican.

Como se puede intuir, para que este tipo de arquitectura sea eficiente es necesario que todos los clientes de la red la implementen y, más aún, obliguen al resto de pares a su uso. Por tanto, para que cualquier transacción tenga lugar entre dos pares de la red, aquel que comparta datos debe necesariamente conocer y validar la identidad de su par así como su reputación, y adicionalmente negar toda operación si dicha reputación no es positiva. De esta forma, el *free-rider* no obtendrá beneficios, ya que rápidamente recibirá una reputación negativa que hará que el resto de pares de la red no compartan contenidos con él.

4.2.1. Identidad digital

Es necesario, como se ha descrito, verificar la identidad de todos los pares de la red en cada momento. Esto comprende dos niveles de confianza. En primer lugar fijaremos

un primer nivel en los esquemas de autenticación que se implementen para verificar la identidad de los usuarios. En segundo lugar, deberemos depositar otro nivel de confianza en los proveedores de identidad que implementarán ese primer nivel.

La solución más intuitiva cuando se trata de implementar la identidad de los usuarios de un sistema consiste en el uso de nombres de usuario y contraseñas asociadas a los mismos. Si bien este tipo de esquema de identidad tiene ventajas muy claras como son la sencillez de uso de cara a los usuarios así como la facilidad de despliegue, no siempre es adecuada en cualquier escenario imaginable. En el caso que nos ocupa, la autenticación debe ser gestionada por el cliente del protocolo *BitTorrent* en representación de su usuario, lo que se conoce como «usuario tras un cliente automatizado». Para implementar una solución basada en contraseñas en este caso de uso, sería necesario bien que el propio cliente almacenase los datos del usuario, bien que se le consultasen al mismo cada vez que se haga uso de la red. El primer caso tiene el inconveniente del almacenaje seguro de la contraseña del usuario, mientras que el segundo es a todas luces poco usable al solicitar interacción al usuario cada vez que éste ejecute el cliente de *BitTorrent*.

Infraestructura de Clave Pública

Más allá de los problemas que podamos encontrar en el lado del cliente, debemos pensar en los que podemos encontrarnos en el lado del servicio. Cada par de la red debe ser capaz de identificar al resto de alguna forma segura, que no implique revelar información de los usuarios. En entornos web, generalmente se realiza la autenticación del usuario mediante su identificador y contraseña asociada, y posteriormente se le entrega una *cookie* que servirá para identificarle en interacciones posteriores. Esta *cookie* no es más que un *token* o seña de identidad aplicada a entornos web. Por desgracia, en el caso que nos ocupa, y pese a que parte del protocolo *BitTorrent* utiliza *HTTP* para la transferencia de información, no es posible el uso de estas *cookies* para garantizar la identidad. Más aún, cualquier *token* que se utilice en su lugar, deberá ser firmado por un tercero de confianza que garantice que esa identidad es correcta y ha sido verificada. Atendiendo a dicho requisito, resulta natural pensar en certificados digitales. Este tipo de certificados se basan en la criptografia de clave pública y la firma digital para asegurar que quien presenta uno de ellos como seña de identidad es quien dice ser, y adicionalmente es posible verificar dicha identidad a través de un tercero en el que se confíe o que se pueda verificar a su vez. La

utilización de formatos universales para la emisión de los certificados, como bien puede ser el estándar `X.509`[x509-rfc], facilita el intercambio de este tipo de información y su uso adecuado para nuestros propósitos.

La criptografia de clave pública o asimétrica[handbook-cryptography] se basa en la utilización de dos claves relacionadas entre si de forma unívoca. Una de ellas se puede hacer pública, mientras que la otra se mantiene en un ámbito estrictamente privado y bajo fuertes medidas de seguridad. La clave pública se obtiene a partir de la clave privada y es la única que se corresponde con ella, de forma que una clave pública se corresponde solamente con aquel que tenga su clave privada asociada. Ambas claves pueden usarse para los procesos complementarios de cifrado y descifrado de información. De esta forma, para descifrar datos previamente cifrados con una de estas claves, es necesario utilizar su clave opuesta. Por ejemplo, si ciframos un conjunto de datos con una clave pública, sólo el poseedor de su correspondiente clave privada será capaz de descifrarlo y por tanto tener acceso a la información, con lo que se consigue la privacidad en las comunicaciones. Si, por contra, se cifra un conjunto de datos con una clave privada, sólo podrá descifrarse con su clave pública asociada, lo que garantiza que sólo el poseedor de dicha clave privada ha podido cifrar dichos datos. Este último proceso, descrito muy a grandes rasgos, es conocido como «firma digital», y permite asegurar una serie de propiedades análogas a una firma tradicional, como por ejemplo la identidad del que firma los datos, así como el «no repudio» o incapacidad del firmante para negar que él mismo ha efectuado la firma. Todas estas propiedades se basan en la garantía ofrecida por la criptografia de clave asimétrica que asegura que es computacionalmente imposible obtener una clave privada a partir de su clave pública correspondiente, así como resulta también inabordable el problema del descifrado un mensaje con una clave que no sea el par correspondiente a aquella con la que se cifró, y por tanto la fuerza bruta es un ataque inútil en este caso.

De este modo podemos avanzar un paso en la elección de las tecnologías utilizadas para la solución aquí propuesta. Si los pares de una red *BitTorrent* intercambian certificados que contengan sus claves públicas y una firma que asegure que son propietarios de los mismos (es decir, que disponen de la clave privada asociada a la clave pública presentada en el certificado) tendrán un mecanismo que les permita identificarse mutuamente de forma unívoca. Sin embargo, esto no garantiza de ningún modo la identidad de los pares, ni evita que éstos generen un nuevo par de claves asimétricas cuando así lo deseen, cambiando por

tanto de identidad. *PGP* (acrónimo de *Pretty Good Privacy*) es una posible solución a este problema, proporcionando repositorios de claves y certificados en los que los usuarios del sistema construyen un modelo distribuído basado en la confianza entre iguales. En el caso que nos ocupa, sin embargo, no es un modelo adecuado ya que supone depender en gran medida de la confianza que pongamos en las firmas que los usuarios realizan verificando sus identidades entre sí. Desgraciadamente, este método no garantiza de ninguna forma que una certificación hecha por un usuario sea fiable, ya que no impone ningún tipo de regla ni restricción. Por contra, otros modelos como los basados en Infraestructura de Clave Pública[pki-rfc] (*Public Key Infrastructure, PKI*), sí definen un conjunto de normas que rigen el proceso de firma en las que podemos basar nuestro modelo de confianza. La *PKI*, al igual que *PGP*, está basada en la firma digital utilizada para certificar que una clave pública se corresponde con una determinada identidad asociada. En este caso, sin embargo, se construye una estructura jerarquizada de entidades, cada una con su propia identidad digital en forma de par de claves asimétricas, que se certifican mediante firmas digitales hasta llegar a una raíz de confianza bien conocida y confiable. Así, el proceso de certificación consiste en la verificación de las firmas efectuadas sobre las claves públicas de aquellos que se certifican, de forma secuencial hasta que una de esas claves públicas presentadas en forma de certificado se corresponda con una conocida previamente y en la que hayamos depositado nuestra confianza. En definitiva, en el caso concreto que nos ocupa es necesario contar con una solución que, tal y como la Infraestructura de Clave Pública permite, garantice que la evaluación del proceso de certificación se pueda realizar de forma automatizada en base a unas reglas preconcebidas, sin la necesidad de la intervención del usuario a la hora de decidir la legitimidad de una identidad o una firma.

Por tanto, llegamos a un primer atisbo de solución que permita identificar a los pares de una red y garantizar dichas identidades a lo largo del tiempo. Será pues necesario construir una Infraestructura de Clave Pública que certifique a los pares de la red, y éstos a su vez deberán contar con un par de claves asimétricas que garanticen su identidad y serán certificadas por la *PKI*. Resulta evidente que las entidades de certificación deben corresponderse con organizaciones del mundo real que se responsabilicen de sus usuarios y garanticen que éstos no pueden cambiar de identidad cuando así lo deseen.

Llegados a este punto, resulta innecesario plantearse el uso de identificadores de usuario y contraseñas, ya que éstos no suponen ninguna medida de seguridad adicional frente a

la solución presentada basada en criptografia de clave asimétrica. De esta forma, además, se evita que el usuario deba introducir su nombre de usuario y su contraseña en ningún momento para usar un cliente de *BitTorrent*, sino que basta con que configure éste con el certificado que habrá obtenido de su organización (aquella que garantiza su identidad y le proporciona, por tanto, acceso a la red) y la clave privada correspondiente.

Identificadores únicos

Una vez solucionado el problema de la identificación de los pares, debemos buscar una forma de referirnos a una identidad concreta a la hora de realizar las operaciones necesarias del protocolo. Si, por ejemplo, deseamos obtener la reputación de un usuario concreto, debemos tener una forma de referirnos a él. Pese a que ya tenemos una seña de identidad o *token* consistente en un certificado digital, éste supone un gran volumen de información que no interesa transmitir por la red con cada operación. Es por tanto que se hace necesario el uso de algún tipo de identificador corto y preciso que permita referirnos a cada nodo que forma parte de la red. Dado que los certificados digitales proporcionan numerosos campos de texto en los que se describen tanto la identidad del certificado como del certificante, resulta inmediato asociar alguno de esos campos como identificador corto del propietario del certificado. El más apropiado de estos campos para el uso que se necesita es el «Nombre Distinguido del Sujeto» o *Subject Distinguished Name*. Este campo hace referencia al beneficiario de la certificación y típicamente se corresponde con un nombre de usuario o una dirección de correo electrónico. Decidido pues dónde almacenar este tipo de identificador, debemos decantarnos por tanto por una convención o norma para codificar el mismo. Los nombres de usuario tienen el inconveniente de que no son únicos, y dos usuarios de dos organizaciones distintas pueden coincidir en el mismo identificador. Las direcciones de correo electrónico, por contra, si se pueden considerar únicas, ya que designan un usuario en una organización, si bien tienen la contrapartida de ser un dato personal que no interesa divulgar. Por tanto se hace necesaria una alternativa, como por ejemplo el «Nombre Uniforme de un Recurso» o *Uniform Resource Name, URN*. Un *URN* es una clase de identificador con un esquema y una sintaxis predeterminada, que permite la persistencia e independencia de la localización de los recursos, y que típicamente les da nombre dentro de la organización a la que pertenecen sin entrar a hacer público ningún dato personal concreto (como ocurre con las direcciones de correo electrónico). Según la definición oficial del estándar[urn-rfc],

un *URN* puede ser algo como, por ejemplo, `urn:es:urjc:etsii:jaime.perez`. Evidentemente, este tipo de identificadores pueden ser ofuscados de forma que no revelen ningún tipo de información, y sólo sirvan para, exclusivamente, referirse a una identidad concreta, por ejemplo `urn:es:urjc:etsii:user3029`.

Gestión de la identidad

Como ya se ha comentado, resulta deseable que diversas organizaciones puedan participar en la red, de forma que no sea gestionada por una única entidad, de tal modo que cada una de ellas pueda llevar la gestión individual del conjunto de sus usuarios. Teniendo en cuenta que es necesario verificar también la identidad de las organizaciones, y no sólo la de los usuarios, es necesario buscar una arquitectura que permita a todas estas organizaciones unirse en la red y establecer lazos de confianza entre si. Los esquemas clásicos de Infraestructura de Clave Pública permiten este tipo de arquitectura, si bien requieren de una raíz en la jerarquía que actúe como certificador de todas las organizaciones participantes. Sin embargo, es una solución poco flexible ya que impone el requisito a cada organización que desee unirse a la red de obtener un certificado emitido por la raíz de confianza común, lo cual puede llevar tiempo y diversas gestiones. Una alternativa cada vez más considerada a este tipo de arquitectura son los servicios de «metadatos», en los que se publica información sobre el sistema. En el caso que nos ocupa, un servicio de metadatos podría almacenar información sobre todas y cada una de las organizaciones participantes, como por ejemplo su raiz de confianza (lo cual permite que cada organización tenga su propia Infraestructura de Clave Pública y por tanto su propia jerarquía) o los proveedores de identidad u otros servicios que proporcione la organización en la red. De este modo, teniendo un servicio de metadatos (que bien pueden ser un conjunto de servidores no centralizados, gestionados por diferentes organizaciones, y configurados a modo de réplica los unos de los otros, al igual que funcionan los servidores raíz del servicio *DNS* en Internet), la inclusión de una nueva organización en la red puede ser tan sencillo como añadirla a los esquemas de autorización que le permitan publicar su propia información en dicho servicio de metadatos.

Para implementar este tipo de servicios es necesario la utilización de algún tipo de lenguaje o convención que permita describir los datos almacenados de forma que su semántica sea clara y reconocible por todos los participantes en la red. En la actualidad, el método más común y aceptado de almacenar datos de forma flexible y extensible son los lenguajes

de marcado, y más en concreto, *XML*[xml-spec] (Lenguaje de Marcado Extensible). Por tanto, podemos incorporar a nuestra arquitectura un servicio de metadatos que utilice un esquema concreto basado en *XML* para definir y almacenar la información de todas y cada una de las organizaciones participantes en nuestra red.

Infraestructuras de Autenticación y Autorización

Si aunamos todo este conjunto de requisitos, a saber, identidad digital basada en certificados, Infraestructura de Clave Pública y un servicio de metadatos, es una buena idea buscar una plataforma ya existente que reuna todas estas funcionalidades y nos permita beneficiarnos de ella a la hora de implementar la arquitectura propuesta sobre el protocolo original *BitTorrent*. En primer lugar, es una buena idea buscar un esquema de *XML* especialmente diseñado para intercambiar el tipo de información que necesitamos en este caso de uso concreto, es decir, identidad digital, autenticación, autorización, e intercambio de atributos relativos a los usuarios. Afortunadamente ya existe un lenguaje con este propósito, llamado *SAML* (*Security Assertion Markup Language*), basado en *XML* y especialmente diseñado para el intercambio de información de seguridad. Más concretamente, la definición del estándar en su segunda edición, *SAML* 2.0[saml-spec], contempla una extensión al esquema para permitir el intercambio de metadatos como aquellos necesarios para el desarrollo de este proyecto.

Dada la idoneidad de este lenguaje, *SAML*, para la resolución del problema que se nos plantea, debemos plantearnos el uso de alguna implementación concreta del mismo en el lenguaje de programación que más nos interese. Debemos considerar así mismo la conveniencia de usar una implementación rasa del lenguaje o una plataforma completa que se sirva de *SAML* para ofrecer servicios de más alto nivel. En el primer caso, podemos considerar *OpenSAML*[opensaml] como una de las implementaciones más extendidas y aceptadas en la actualidad. Su principal desventaja es que obliga a utilizar el lenguaje a un bajo nivel y por tanto entrar en demasiados detalles que no necesariamente tienen por qué ser necesarios para nuestros propósitos. Por contra, *Shibboleth*[shibboleth] es una plataforma completa desarrollada por la Iniciativa por el «Middleware» de Internet2[2] que

[2]Internet2 es un consorcio sin ánimo de lucro que desarrolla aplicaciones y tecnologías avanzadas de red para el académico. Actualmente proporciona la red dacadémica norteamericana y satisface sus necesidades y requisitos de uso intensivo de ancho de banda.

proporciona servicios de alto nivel como *Single Sign On* basados en el uso del lenguaje *SAML*. Curiosamente, *Shibboleth* está implementado haciendo uso de *OpenSAML* y su popularidad ha crecido ostensiblemente en los últimos tiempos.

Si bien *Shibboleth* parece una buena alternativa, basada en *software libre*, existen otras que podrían ser interesantes para nuestros propósitos. En un primer lugar, podemos considerar el software *simpleSAMLphp* escrito en el lenguaje *PHP*, que facilita gracias a su sencillez el despliegue y desarrollo de aplicaciones compatibles con el lenguaje *SAML* y por ende con *Shibboleth*. *simpleSAMLphp* es un desarrollo del departamento de investigación y desarrollo de la red académica noruega, *UNINETT*[uninett], y ha obtenido un gran éxito desde su reciente aparición, precisamente gracias a la sencillez de la que hace gala. Desgraciadamente tiene aún algunas limitaciones propias de su corta existencia en ámbitos que resultan básicos para el desarrollo de este proyecto, como por ejemplo la firma y validación de certificados digitales y documentos *SAML*.

Por último analizamos la tercera plataforma candidata, en la que se basa, de hecho, la implementación ya presentada de *simpleSAMLphp*. Hablamos de *eduGAIN*[edugain], una infraestructura de autenticación y autorización (*AAI*) que proporciona servicios de identidad digital de alto nivel sobre la base de la librería *OpenSAML*. *eduGAIN* es un desarrollo de la red académica paneuropea cuyo objetivo es proporcionar la interoperabilidad entre otras infraestructuras de autenticación y autorización existentes, en lo que se ha dado en denominar como «confederación». Es por tanto compatible de forma directa con *Shibboleth* y permite añadir de forma sencilla una gran cantidad de tecnologías de federación a la red ya existente, así como nuevos perfiles que definan distintos protocolos e intercambios de mensajes entre entidades que formen parte de la arquitectura, motivo por el cual *eduGAIN* resulta idóneo para el desarrollo de este proyecto. Más aún, la participación dentro del propio proyecto *eduGAIN* facilita la capacidad para extenderlo adecuándose a las necesidades que requiera el desarrollo de este proyecto, así como adaptar su modelo de confianza[edugain-cookbook], ya de por si muy similar al de la solución descrita.

4.2.2. Reputación persistente

Una vez solucionados todos los aspectos relativos a la identificación de los pares de la red de forma unívoca y a lo largo del tiempo, nos aprovecharemos de ello para asociar

a cada usuario de la red una reputación, o lo que es lo mismo, una descripción de su comportamiento para con el resto de pares en cada una de sus interacciones. Por ello necesitamos que esta reputación se mantenga de forma persistente en el tiempo y se pueda consultar y actualizar independientemente de la ubicación tanto del usuario que la solicita o reporta, como del objeto de dicha reputación.

Dado que la arquitectura propuesta implica la gestión de las identidades por parte de entidades u organizaciones que se responsabilicen de las mismas, parece lógico delegar también la gestión y almacenaje de la reputación de cada usuario en manos de las mismas. Por tanto debemos valorar la forma de almacenar dicha reputación, así como el interfaz que deseamos ofrecer tanto para consultarla como para actualizarla en base a una interacción reciente. En ese sentido disponemos de dos alternativas diferentes a la hora de abordar este problema:

- Por un lado, podemos diseñar un servicio nuevo de consulta y notificación de reputaciones, que sea centralizado a cada organización (es decir, cada una podrá disponer de este servicio, replicado o no), que sea el encargado de, mediante un interfaz a definir, recibir y responder tanto las peticiones de reputación de un usuario como las notificaciones. Adicionalmente, deberán servir de interfaz de cara al servicio de autenticación de la organización, de forma que sean capaces de traducir desde el protocolo diseñado para el caso que nos ocupa a otros protocolos como pueden ser *LDAP*[ldap-rfc] («Protocolo Ligero de Acceso a Directorios»), o *RADIUS*[radius-rfc] («Servicio de Autenticación Remota de Usuarios por Marcación»).

- Por otro lado, existe la posibilidad de usar los «rastreadores» o *trackers* que ya están definidos por el propio protocolo *BitTorrent* como puntos en los que cada organización podrá gestionar las consultas y notificaciones de reputación de sus usuarios. En tal caso, sería necesario extender el protocolo utilizado por dichos *trackers* (basado en *HTTP*) para que soporte las operaciones necesarias, así como para servir de pasarela frente a los protocolos ya existentes en la organización, como se ha expuesto en el caso anterior. De esta forma, cada organización participante de la red debería instalar al menos un servidor en el que se podrían dar dos servicios de forma conjunta, el de *tracker* del protocolo original y el de gestión de la reputación, con el consiguiente ahorro de recursos y la sencillez en el despliegue que ello supone.

Evidentemente, en términos de esfuerzo y cantidad de trabajo a realizar, así como de aprovechamiento de esfuerzos ya realizados por otros, el segundo caso es más adecuado tanto para un hipotético despliegue de la red aquí propuesta como para su propia implementación. Esto significa que será por tanto necesario modificar un *tracker* del protocolo *BitTorrent* para que además de sus tareas habituales sea capaz de gestionar nuevas operaciones relativas a la reputación.

Por supuesto, resulta necesario en cualquier caso, modificar también un cliente *BitTorrent* de forma que realice consultas y notificaciones de reputación al *tracker* adecuado cada vez que surja una transacción con otro par de la red. El hecho de aprovechar los propios *trackers* facilita también esta tarea, puesto que sólo es necesario extender el protocolo, y no incluir nuevos tipos de conexiones ni de nodos en la red. Así, se facilita sobremanera la tarea de un cliente de la red, que deberá consultar la reputación de un par cada vez que éste le solicite una transferencia, además de notificar su comportamiento al solicitarla él mismo y en base a la respuesta recibida (que, por supuesto, puede ser negativa).

Es necesario así mismo prestar atención a un aspecto importante en una arquitectura tan distribuída como la propuesta, como es el de la localización de los recursos. Ya que cada organización será capaz de utilizar su propia Infraestructura de Clave Pública y poner en servicio sus propios *trackers* y servicios de gestión de reputación, es necesario a su vez una infraestructura que permita localizarlos en la red, independientemente de la ubicación de los pares. Afortunadamente ya contamos con un método eficaz para resolver este problema gracias al servicio de metadatos. Ya que para la gestión de la identidad digital almacenaremos datos de las organizaciones en servidores de metadatos bien conocidos, es trivial aumentar la cantidad de información almacenada para incluir la localización de servicios adicionales proporcionados por éstas. Además, mediante el uso de *URNs* es posible conseguir la identificación de una organización concreta de forma corta y sencilla, y en base a ella realizar búsquedas en el servicio de metadatos que nos permitan obtener toda la información completa relativa a la organización y establecer transacciones con sus usuarios.

En este caso es necesario tener en cuenta que los identificadores de cada organización deben ser únicos, lo que implica la obligatoriedad de disponer de medidas de seguridad específicas que garanticen la unicidad de los mismos. Una forma sencilla de conseguir esto es la inclusión de un servicio de registro de identidades asociado a la arquitectura

y totalmente independiente de las organizaciones participantes, al cual deberán acudir éstas para formalizar el registro de sus propios identificadores y garantizar de ese modo la imposibilidad de suplantar la identidad de cualquier otra organización, así como mantener una coherencia en la normativa de nombrado e identificación.

El mayor problema a solucionar en este caso es la distribución del mencionado identificador de las organizaciones para que los clientes de la red puedan consultarlo y referirse a ellas en el servicio de metadatos. La solución es intuitiva, y coincidente con los identificadores de los propios clientes: basta incluir estos identificadores de organización en cada certificado emitido por las mismas. De esta forma, cuando un cliente del sistema quiera consultar la reputación de otro par, no tendrá más que obtener su certificado, verificar su firma, y en base a ello lanzar una consulta para obtener la reputación del par identificado por el *Subject Distinguished Name* del certificado, perteneciente a la organización identificada por otro campo incluido en el mismo certificado de usuario, como bien puede ser el *Subject Alternative Name* proporcionado por la extensión del mismo nombre definida en el estándar `X.509`.

Por último, queda pendiente la cuestión de la forma de representar la reputación de cada par de la red. Si bien este aspecto queda fuera de los objetivos iniciales de este proyecto, en cualquier caso es necesario abordar el problema al menos de una forma básica que permita llegar a una demostración sencilla de la solución propuesta. Por ello la representación elegida se reduce a una simple oscilación entre valores negativos (mala reputación) y positivos (buena reputación, descriptiva de un comportamiento adecuado) cuyo valor absoluto no excede en ninguno de los casos de uno. Dicha reputación se verá aumentada al notificar un par una transacción realizada con éxito, y decrementada en caso contrario. En base a dicho valor numérico los pares deberán decidir si bloquean o no a otros pares, en cuyo caso nos limitaremos a desbloquear a aquellos pares con una reputación mayor que cero y bloquear al resto.

El planteamiento propuesto, pese a su extrema sencillez, nos permitirá solventar el problema del *free-riding*, siempre y cuando se cumplan dos condiciones concretas de partida:

- Todos los pares se inician en la red con un certificado y una reputación positiva mínima. Esto permitirá que otros pares compartan contenidos con el recién llegado y así éste podrá solucionar el problema de la entrada inicial en la red o *bootstrapping*.

- En segundo lugar, al iniciarse con una reputación mínima, los *free-riders* sólo podrían beneficiarse de la transferencia de un volumen ínfimo de datos, ya que tan pronto como reciban una notificación negativa su reputación se verá decrementada hasta valores nulos o negativos que no les permitirán continuar con la descarga. De este modo, si los contenidos publicados en la red se dividen obligatoriamente en más de un «pedazo»[3], será necesaria más de una transferencia para obtener un archivo en su totalidad, siendo esto imposible en caso de que el par no tenga un comportamiento adecuado con el resto de la red.

Pese a estas dos presunciones, no es difícil darse cuenta de que cuando un *free-rider* no reciba notificaciones de reputación negativas, su reputación no se verá afectada y podrá seguir beneficiándose del resto de usuarios de la red. Por ello, es fundamental que todos los pares que la integran notifiquen la reputación de aquellos con los que inician una transacción siempre que esto ocurra. Si esta condición se cumple, el único caso en el que un *free-rider* no vería mermada su reputación es aquel en el que ningún otro par de la red le solicite transferencias de datos. Por supuesto, es imposible determinar el comportamiento de un nodo en base a algo que no ha sucedido, y por tanto no se puede hacer nada para evitarlo. Más aún, en un caso como el descrito, el que un nodo se niegue a compartir con otros nodos no supone un problema mientras nadie le solicite transferencias, ya que no se está produciendo ningún perjuicio real a la red, y por tanto tal escenario entra dentro del funcionamiento normal de una red de pares.

4.2.3. Mecanismos de seguridad

En un sistema de reputación como el aquí descrito existe la posibilidad abierta de distintos tipos de fraude o de ataques contra el mismo, por lo que es necesario tomar medidas a nivel de los protocolos y comportamientos de cada elemento del sistema. Este tipo de fraudes se reducen, en el caso que nos ocupa, a notificaciones de reputación con

[3]En la definición oficial del protocolo *BitTorrent*, un «pedazo» es cada una de las partes iguales en las que se divide un fichero publicado para facilitar su transferencia, generalmente de 256 kilobytes de tamaño. Cada uno de dichos «pedazos» tiene su correspondiente resumen de 20 bytes calculado mediante el algoritmo *SHA1* en el «torrent», y las peticiones realizadas entre los pares hacen referencia a dichos «pedazos», de forma que al terminar cada transferencia parcial es posible verificar la integridad de los datos recibidos.

ánimo de lucro propio o de perjuicio a un tercero. En concreto, podemos observar dos tipos de ataques:

- Notificaciones de reputación a la baja. En este caso, un nodo de la red realiza notificaciones negativas relativas a otro nodo, sin que medie transacción previa de ningún tipo entre ambos, con el único fin de perjudicar a dicho nodo.

- Notificaciones de reputación al alza. En tal caso, dos o más nodos de la red pueden llegar a un acuerdo mediante el cual se realizan notificaciones de reputación positivas entre ellos de forma que su reputación aumente sin necesidad de la existencia de un intercambio de información.

En ambos casos es factor común la ausencia de intercambio de información como suceso que provoque una notificación de reputación. Para solucionar este problema, se implementa una primera medida en forma de identificadores aleatorios de transacciones. Cada intercambio de información (que, además, es direccional de un par a otro) lleva asociado un identificador numérico aleatorio. Cuando un par solicita una transacción al cliente, éste le informa del identificador de transacción que debe utilizar. El cliente, además, informa a su *home tracker* del identificador de transacción al realizar la consulta de reputación del par mediante el pertinente protocolo. Cuando el par realiza una notificación de reputación al *home tracker* del cliente, debe presentar su propia identidad así como el identificador de transacción obtenido del cliente. De este modo, cualquier notificación que no vaya acompañada de un identificador de transacción recibido previamente en una operación de consulta, o en la que no coincida la identidad del notificante con la de aquel cuya reputación se solicitó para la transacción a la que se hace referencia, será automáticamente denegada y desechada por parte de los *trackers*.

Huelga decir, por supuesto, que toda notificación de reputación de un par sobre si mismo está totalmente prohibida y debe ser denegada por su *home tracker*, dejando en manos de la implementación concreta del *tracker* si tal comportamiento es penalizado en términos de reputación o de cualquier otra forma.

Por otro lado, existe el problema de la evaluación de las notificaciones en si mismas, aún en el caso en el que sí se refieran a una transacción legítima. Existen dos aproximaciones que permiten abordar semejante problema:

- Utilizar una nueva reputación, relativa exclusivamente a la fiabilidad de las notificaciones de aquel que la ostenta. Esto implica la construcción de un meta sistema de reputaciones que se valore a si mismo, con el aumento por tanto de complejidad tanto a nivel de protocolos como de implementación.

- Por otro lado, cabe la posibilidad de utilizar la propia reputación de cada nodo para valorar la fiabilidad de sus notificaciones. En tal caso se hace necesario ponderar las reputaciones notificadas con la propia reputación del notificante. Esto implica simplemente un aumento de cálculo pero no ninguna modificación al protocolo, ya que los notificantes incluyen un documento con su identidad y su propia reputación firmado por su *home tracker*.

La segunda opción descrita es óptima para la resolución del problema en el caso que nos ocupa, ya que es considerablemente más sencilla y no obstaculiza ningún mecanismo implementado por los protocolos aquí descritos.

Mediante estas sencillas medidas es posible aumentar significativamente la fiabilidad de las notificaciones que tienen lugar en el sistema y reducir el impacto de posibles ataques de fraude, organizados o no, que se realicen contra él.

Sin embargo, queda un último aspecto a considerar en cuanto a posibles usos fraudulentos del sistema. Es el caso de *trackers* que manipulen de cualquier forma las reputaciones de los pares de los que son responsables, ya sea para obtener un beneficio o para causar un perjuicio concreto. Este tipo de ataque debe ser analizado a posteriori, ya que no hay forma de garantizar el comportamiento de un *tracker* acorde con las normas definidas por la arquitectura. Para proceder a evaluar dicho comportamiento resultan necesarias una serie de medidas:

- En primer lugar, debe ser requisito obligatorio que los *trackers* del sistema almacenen un registro de actividad completo en el que queden reflejadas tanto las peticiones como las notificaciones de reputación recibidas, así como sus respuestas asociadas y los efectos causados -por ejemplo, el cambio de una reputación en un par del cual el *tracker* es responsable.

- Partiendo de lo anterior, es posible requerir a un *tracker* su registro de actividad, que permita evaluar su comportamiento en cualquier momento, en base a una posible

sospecha o a un procedimiento rutinario. Este tipo de comprobación se ha de realizar por una autoridad centralizada encargada del mantenimiento y actualización de los servicios de metadatos. Basándonos en la flexibilidad que proporcionan estos servicios, es posible tomar medidas contra una organización cuyos *trackers* se comporten de forma inadecuada simplemente eliminando su información del servicio de metadatos y bloqueando su acceso al mismo, lo que equivale en la práctica a su salida de la red.

- Por último, como es lógico, es necesario proveer mecanismos de contacto a los usuarios del sistema (bien pueden ser organizaciones, bien usuarios finales de la red) que les permitan reportar posibles abusos.

Todo esto implica, por tanto, la definición de unas políticas de uso claras y estrictas a las que han de adherirse las organizaciones participantes, así como la creación de una autoridad que gestione los servicios comunes de la red y se encargue de la verificación y aplicación de dichas políticas.

4.3. Especificación

A lo largo de la presente sección se describe y detalla la arquitectura propuesta, en términos del modelo de confianza en el que se basa y los protocolos que la componen, más allá del protocolo original *BitTorrent* y sus extensiones oficiales.

4.3.1. Modelo de confianza

En primer lugar es necesario definir el modelo de confianza en el que se asienta la arquitectura aquí descrita y que supone la base teórica del conjunto de protocolos que la componen. Dicho modelo es muy similar al utilizado por *eduGAIN* y está basado en el uso de una Infraestructura de Clave Pública, permitiendo en cualquier caso que cada organización que se una a la red disponga de su propia infraestructura, así como en un servicio de distribución de metadatos que se encarga de la publicación de toda la información relativa a cada organización participante necesaria para el correcto funcionamiento de la red y sus protocolos. De forma adicional al servicio de metadatos, y para garantizar la correcta identificación de todos los integrantes de la red, se incluye un servicio para el

registro de identificadores (basados en el estándar *URN*[urn-rfc]) al que deberá acudir toda organización que desee emitir un certificado nuevo en su *PKI* para el adecuado registro del identificador asociado al mismo. Por conveniencia y comodidad de las organizaciones, este servicio de registro admite la delegación de espacios de nombres completos a las mismas, de forma que éstas puedan gestionar de forma independiente los identificadores registrados bajo su propio dominio. Por ejemplo, si partimos de un espacio de nombres común a todos los participantes `urn:aretusa`, una organización participante podría solicitar la gestión por su propia cuenta de todos los identificadores bajo el espacio de nombres `urn:aretusa:organizacion`.

Así pues, cada organización podrá construir su propia *PKI* o utilizar una ya existente y ligeramente adaptada al escenario que nos ocupa, con sus propias autoridades de certificación (*CA*, de *Certification Authority*) y políticas internas. Cualquier organización que desee adherirse a la red «aretusa» aquí descrita debe, por tanto, asegurarse de que cumple una serie de requisitos técnicos:

- En primer lugar, debe obtener un identificador válido y único en la red mediante el servicio de registro de identificadores. De forma opcional, puede solicitar la delegación del espacio de nombres coincidente con dicho identificador de la organización para la gestión independiente del mismo.

- Debe proporcionar una Infraestructura de Clave Pública que le permita emitir certificados para sus usuarios con unas característica concretas. Los certificados deben seguir la sintaxis indicada por el estándar `X.509`.

- Debe disponer de, al menos, una autoridad de certificación (*CA*) con un certificado autofirmado que debe cumplir todos los requisitos aquí expuestos, o bien adherirse a alguna ya existente. Dicho certificado será incluído como parte de los metadatos correspondientes a la organización en el servicio de metadatos.

- Todos los certificados emitidos por la *CA*, incluído el suyo propio, deben disponer de un identificador único y acorde con el estándar *URN*, indicado en el campo *Subject Distinguished Name* del certificado.

- Todos los identificadores utilizados en certificados deben haber sido previamente registrados en el servicio de registro de identificadores.

- El certificado autofirmado por la autoridad de certificación raíz tendrá como *Subject Distinguished Name* el propio identificador único de la organización.
- Cualquier otra autoridad de certificación deberá ser firmada por la autoridad raíz. Su identificador incluído en el *Subject Distinguished Name* de su certificado será distinto del identificador de la organización e identificará a la autoridad unívocamente.
- Todos los usuarios de la organización que deban acceder al servicio dispondrán de su propio certificado firmado por una autoridad de certificación válida de la misma, cuyo *Subject Distinguished Name* se corresponderá con su identificador único.
- Todos los certificados de la *PKI* implementarán la extensión *Subject Alternative Name* del estándar `X.509` e incorporarán en ella el identificador de la organización, que se corresponde con el espacio de nombres de todos los identificadores pertenecientes a la misma.

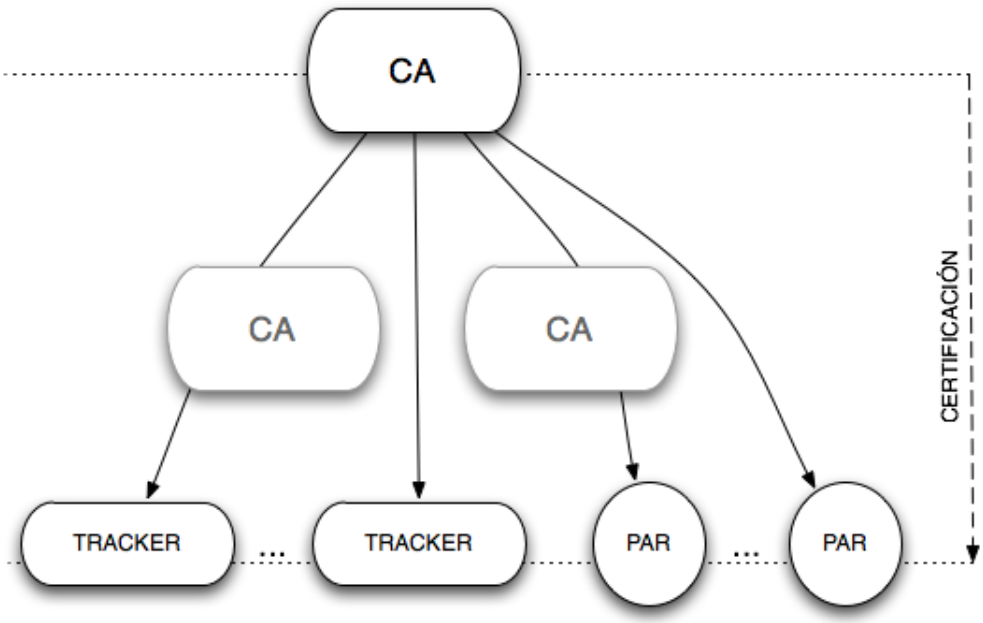

Figura 4.1: Esquema organizativo de una *PKI* en la arquitectura propuesta.

La codificación de los identificadores de cada nodo participante de la red debe sufrir una conversión desde su valor registrado en forma de *URN* al formato en el que se almacenará en los campos *Issuer Distinguished Name* y *Subject Distinguished Name* de los

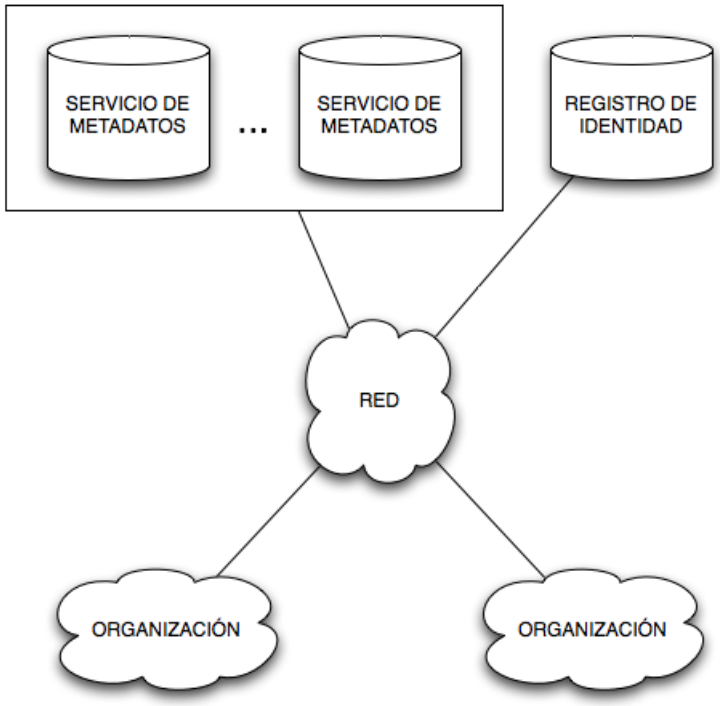

Figura 4.2: Esquema de la arquitectura «aretusa».

certificados, conforme al estándar `X.500`[x500-rfc]. Para ello se descompondrá el identificador en tantos elementos como partes sean resultantes de su descomposición atendiendo al carácter "`:`", eliminando la cadena `urn` inicial. De los elementos restantes se asignarán de izquierda a derecha y en el mismo orden cada uno de ellos al valor de un atributo de tipo `DC`, asignando al último elemento del conjunto el «nombre común» o *Common Name* (*CN*) del *Subject Distinguished Name* o *Issuer*, según corresponda. Por ejemplo, la conversión del *URN* de una organización `urn:aretusa:organizacion` tendrá su equivalente para su inclusión en el *Subject Distinguished Name* de su autoridad raíz de certificación en la forma `DC=aretusa, CN=organización`. Para una organización como por ejemplo la Universidad Rey Juan Carlos, su identificador asociado tendrá la forma `urn:aretusa:es:urjc`, mientras que los *Distinguished Name* construidos a partir de dicho identificador serán `DC=aretusa, DC=es, CN=urjc`. Partiendo de esta base, la organización podrá pedir la delegación de su espacio de nombres y aplicar las convenciones y normas que considere oportunas. Por ejemplo, si la Universidad Rey Juan Carlos desea distinguir a sus usuarios en función de si son empleados o alumnos, podría definir a un alumno con el identificador

`urn:aretusa:es:urjc:alumnos:nombre` y su correspondiente *DN* `DC=aretusa, DC=es, DC=urjc, DC=alumnos, CN=nombre`, donde `nombre`, por supuesto, podría ser cualquier cadena única definida por la organización que le permita identificar a un usuario concreto, desde el nombre y apellidos del usuario, hasta una secuencia aleatoria de caracteres.

Adicionalmente, y como ya se ha expuesto, todos los certificados correspondientes a una misma organización deben incluir la extensión *Subject Alternative Name* del estándar `X.509`. Dicha extensión soporta un número indeterminado de campos de este tipo que contienen un atributo con un tipo predefinido por el estándar, a elegir entre un conjunto finito. Dadas las limitaciones de dicho conjunto de tipos, y por compatibilidad con el modelo de confianza utilizado en *eduGAIN*, se utilizará un atributo de tipo *URI* o *Uniform Resource Identifier* (Identificador Uniforme de Recurso)[uri-rfc], que contendrá una dirección fija y bien conocida a la que se concatenará el identificador que se desea incluir en el campo, codificado según especifica el estándar *URL*[url-rfc]. De este modo, el campo *Subject Alternative Name* incluído en todos los certificados pertenecientes a una organización será `http://registry.aretusa.org/resolver?urn=urn\%3Aaretusa\%3Aorganizacion`, como por ejemplo `http://registry.aretusa.org/resolver?urn=urn\%3Aaretusa\%3Aes\%3Aurjc` en el caso de la Universidad Rey Juan Carlos.

4.3.2. Protocolo de autenticación entre pares

Independientemente de la infraestructura inherente a la arquitectura aquí propuesta, es necesario que los clientes de *BitTorrent* implementen una serie de protocolos adicionales al original. El primero y más sencillo de ellos, pero no por ello menos importante -de hecho su relevancia es fundamental para el correcto funcionamiento del sistema-, es el Protocolo de Autenticación entre Pares (PAP) o intercambio de certificados. Este protocolo permite a dos pares cualesquiera de la red intercambiar sus respectivos certificados al inicio de una transacción, de forma que puedan autenticase mutuamente y verificar sus identidades, así como utilizar dichas identidades para efectuar consultas o notificaciones de reputación.

El intercambio de certificados se realiza entre dos pares que establecen una conexión con objeto de realizar alguna transferencia de datos. Dicho intercambio ocurre tras el establecimiento completo de la conexión y la negociación de extensiones, tal y como se presenta en la figura 4.3, y se basa en el protocolo de extensión de *Azureus* para facilitar

la implementación. Por tanto, un cliente cualquiera de *BitTorrent* que no sea el propio *Azureus* deberá soportar el Protocolo de Mensajería de Azureus (*AZMP*) [4] y el mensaje adicional introducido por el protocolo de autenticación de pares de «aretusa». Como se describe en la figura, un cliente «aretusa» deberá ser capaz de intercambiar el siguiente conjunto de mensajes con otros pares:

1. Un par envía al cliente un *handshake*[5] *BitTorrent*, indicando que soporta el Protocolo de Mensajería de Azureus. Como ya se ha comentado anteriormente, es deseable que además soporte el Protocolo de Negociación de Extensiones, aunque esto no es estrictamente necesario para el funcionamiento general del protocolo aquí expuesto.

2. El cliente responde al par con otro *handshake BitTorrent* en el que indica el conjunto de extensiones que soporta, entre las cuales debe estar presente de forma obligatoria el Protocolo de Mensajería de Azureus, al igual que en el caso del par.

3. El par recibe la respuesta del cliente y al comprobar que éste soporta el protocolo *AZMP*, le envía un *handshake* en el que debe indicar soporte del protocolo «aretusa» y más concretamente del mensaje `ARETUSA_EXCHANGE`.

4. El cliente contesta al *handshake AZMP* con el suyo propio, indicando soporte del protocolo «aretusa».

5. Finalizada la negociación inicial del Protocolo de Mensajería de Azureus, comienza realmente el Protocolo de Autenticación entre Pares. El par construye un mensaje de intercambio de «aretusa» cuyo contenido es un diccionario *BitTorrent*[6] con los campos «cert», que contiene el certificado del par; «signature», que contiene el *Subject Distinguished Name* del certificado firmado por la clave privada del par y «transaction», una sucesión aleatoria de dígitos que identifica la transacción actual.

[4] La descripción técnica del Protocolo de Mensajería de Azureus queda fuera del ámbito de este proyecto, por lo que se recomienda la lectura de la especificación de dicho protocolo[azmp-spec].

[5] *Handshake* es la denominación habitual del mensaje o conjunto de mensajes que forman parte del proceso automatizado de negociación que establece dinámicamente el conjunto de parámetros que rigen un canal de comunicaciones.

[6] En *BitTorrent*, las estructuras de datos denominadas «diccionarios» son diccionarios tradicionales construídos con una codificación especial denominada *bencoding*, descrita ampliamente en la especificación no oficial del protocolo *BitTorrent*[bt-spec-nonofficial]

6. El cliente construye un mensaje de intercambio de «aretusa» de forma idéntica al par y lo enviará a modo de respuesta, terminando el intercambio de certificados que permitirá a ambos identificarse mutuamente.

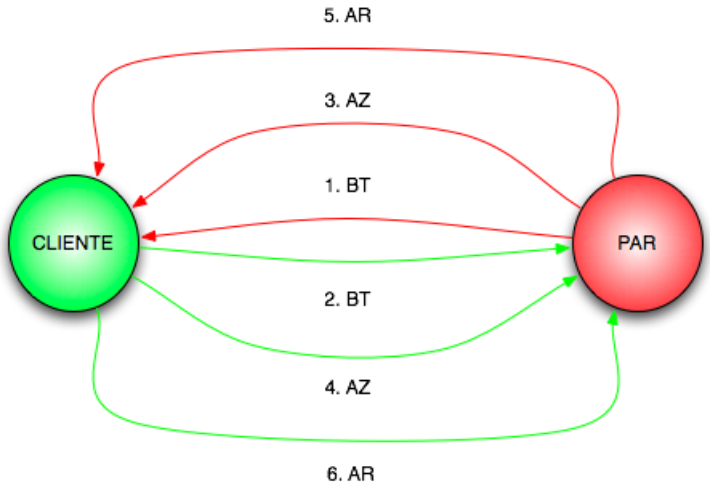

Figura 4.3: Protocolo de autenticación entre dos clientes de la red «aretusa».

Tanto el cliente como el par deben realizar una serie de verificaciones básicas sobre la información recibida en el mensaje de intercambio de «aretusa». A grandes rasgos, es necesario verificar la firma recibida con el propio *Subject Distinguished Name* del certificado, en cuyo caso se tendrá certeza de que el nodo al otro extremo se encuentra en posesión de la clave privada asociada a la pública almacenada en el certificado. Así mismo, resulta recomendable extraer el identificador de la organización que ha emitido el certificado por si coincide con la propia, pudiendo realizar de este modo una validación del mismo frente a la cadena de certificación almacenada localmente, si bien esta es una medida adicional que no resulta necesaria para el correcto funcionamiento del sistema.

En caso de error en la verificación de la firma recibida frente al *Subject Distinguished Name*, el nodo debe cerrar de forma inmediata la conexión y marcar a su contrario como ignorado para garantizar que no se le tiene en cuenta en futuros procesos de desbloqueo optimista[7].

[7] *BitTorrent* define varios posibles estados para un par. Por un lado, puede encontrarse bloqueado (*choked*) o desbloqueado (*unchoked*). Por otro lado, puede ser ignorado (*snubbed*) o no. Sólo cuando un par se encuentre en estado «ignorado» será excluido del proceso periódico de desbloqueo aleatorio, conocido como *optimistic unchoke*.

4.3.3. Protocolo de consulta de reputación

El mero hecho de haber recibido un certificado por parte de un par no garantiza de ningún modo la identidad del mismo, y tampoco proporciona ningún tipo de reputación. Para verificar la identidad de un par que desea una transferencia de bloques con el cliente y obtener información acerca de su comportamiento previo en la red son necesarias dos comprobaciones:

- Verificar que el certificado recibido ha sido emitido por un *tracker* confiable (autenticación basada en criptografía de clave pública). La arquitectura propuesta (similar a la arquitectura de *eduGAIN*) posibilita que un par presente un certificado de su *home tracker* en la red de un *remote tracker*, de modo que es necesario proporcionar un mecanismo que independice a los pares de su *home tracker*, siendo éste localizable desde cualquier otra red a la que puedan conectarse sus usuarios.
- Obtener del *tracker* adecuado la reputación del par. Los clientes deben dirigir sus consultas de reputación al *home tracker* del par que solicita una interacción.

De este modo, cada vez que el cliente reciba un certificado de un par que desea descargar contenidos de él, deberá seguir el siguiente procedimiento:

1. Realizar una petición de reputación sobre dicho par al *remote tracker* (esto es, el *tracker* al que se encuentra conectado actualmente). Dicha petición indicará cuatro parámetros de forma obligatoria: el identificador del par en su organización o *peer id* (coincidente con el *Subject Distinguished Name* presente en el certificado `X.509`), el identificador de la organización a la que pertenece el par o *peer home id* (campo *Subject Alternative Name* en el certificado del par), el identificador de la organización a la que pertenece el cliente o *home id* (campo *Subject Alternative Name* en el certificado del cliente) y finalmente el identificador de la transacción o *transaction*, mencionado por el par en su mensaje de intercambio de certificados. De este modo el cliente realizará una petición `HTTP` codificada de la siguiente forma:

   ```
   http://remote_tracker/reputation?home_id=urn%3Amyhome&peer_home_id=urn%
   3Apeerhome&peer_id=urn%3Apeerhome%3Apeer&transaction=transaction_id
   ```

   Nótese el uso de la codificación definida por el estándar *URL* para la representación de cada uno de los identificadores o *URN* como parámetros de la petición resultante.

2. El *remote tracker* verificará si el cliente que le solicita reputación forma parte de su organización, comprobando el campo *home_id* de la petición. Si no existe coincidencia, buscará el identificador en su memoria caché local (de implementarla) o consultará al servicio de metadatos. En cualquiera de ambos casos, obtendrá información sobre la organización a la que pertenece el cliente, incluyendo la dirección de su *home tracker*, al que redirigirá al cliente mediante una respuesta `HTTP` con código de error 302 (*Found*), cuyo contenido será una respuesta de autenticación de *eduGAIN*. Dicha respuesta contendrá tan sólo una aserción con una `AuthenticationStatement` y resultado `RedirectUserTo`. En caso de error, la respuesta consistirá en el código de error `HTTP` apropiado junto con una respuesta de autenticación de *eduGAIN* con el resultado descriptivo del mismo.

3. El cliente redirigirá su consulta a su propio *home tracker* por medio de la redirección obtenida del *remote tracker*.

4. El *home tracker* del cliente verificará gracias al servicio de metadatos la identidad de la organización del par, y obtendrá la dirección del *home tracker* del mismo de igual forma al caso anterior. Finalmente, y tras haber comprobado que el cliente pertenece a su misma organización, anotará por tanto el identificador de transacción recibido en una tabla interna con un tiempo de caducidad predeterminado. Una vez más, la respuesta será un código de error `HTTP` 302, pero en este caso irá acompañada de una respuesta de autenticación *eduGAIN* de resultado `ConnectTo` y cuyo `AuthenticationStatement` hace referencia al *peer home id*, al igual que una segunda aserción con un *AttributeStatement* que adicionalmente contendrá un atributo con la cadena de certificación del *home tracker* del par. De esta forma, y dado que ambas aserciones irán firmadas por el *home tracker* del cliente, éste puede verificar la confianza en el *home tracker* del par, entendiendo que es un *tracker* conocido y confiable en el sistema y con el que su organización permite la interacción (en caso de no existir políticas que la desautoricen de forma explícita).

5. Por último, se dirige de nuevo la misma petición al *home tracker* del par, incluyendo exactamente los mismos parámetros.

6. En base a la petición recibida, el *tracker* podrá verificar que le corresponde responder

a dicha consulta y el usuario sobre el que se pregunta es un usuario válido de su organización, además de comprobar a qué organización pertenece el cliente y decidir en base a eso y a sus políticas internas si se responde o no a la petición. En caso afirmativo, devolverá una respuesta `HTTP` 200 (*OK*) acompañada de una respuesta de autenticación *eduGAIN* de resultado `Accepted` y con una aserción firmada, que contendrá un `AuthenticationStatement` referenciando el identificador del par sobre el que se realiza la consulta o *peer id* y un `AttributeStatement` con la reputación solicitada del mismo.

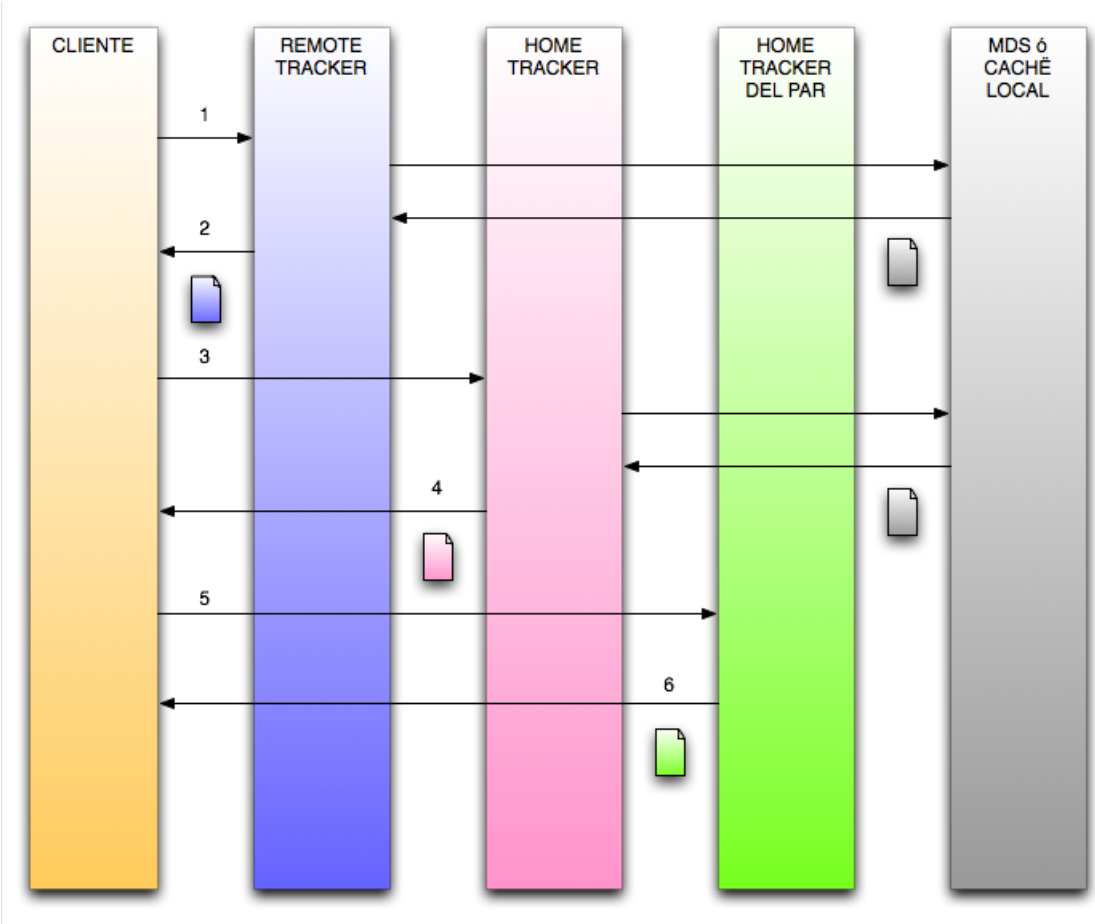

Figura 4.4: Diagrama de una petición de reputación.

En la figura 4.4 se muestra el diagrama que describe el proceso seguido por cada elemento del sistema en una petición de reputación, acorde con cada uno de los pasos presentados. Las respuestas que incluyen un documento *SAML* se representan con el icono de una hoja

de papel, de forma que se observa que todos los mensajes intercambiados por los elementos del sistema incluyen este tipo de documentos salvo las peticiones realizadas por el cliente y los *trackers*.

En caso de que el proceso finalice correctamente, el cliente no tiene más que obtener la lista de atributos contenidos en el `AttributeStatement` del documento *SAML* recibido como respuesta y extraer de ella el atributo `reputation`, cuyo valor será numérico. El comportamiento una vez obtenido y evaluado el valor correspondiente a la reputación del par consiste en ignorarlo (marcarlo, según la nomenclatura de *BitTorrent*, como *snubbed*) si éste no es mayor que cero, y seguir adelante con la transacción en caso contrario.

4.3.4. Protocolo de notificación de reputación

Las notificaciones de reputación se producen cuando el cliente ha comprobado el comportamiento positivo o negativo de un par, lo que quiere decir que dicho par ha compartido información con él o se ha negado a hacerlo pese a habérsele solicitado explícitamente. En tal caso el cliente iniciará el proceso de notificación de forma similar al de consulta:

1. En primer lugar, el cliente dirigirá una notificación de reputación al *remote tracker* añadiendo un quinto parámetro adicional a aquellos que ya se incluían en la consulta, con un valor numérico entero descriptivo del comportamiento observado del que se desea dejar constancia:

   `http://remote_tracker/reputation?home_id=urn3Amyhome&peer_home_id=urn%3Apeerhome&peer_id=urn%3Apeerhome%3Apeer&transaction=trans_id&reputation=X`

   Donde `X` es el valor entre -1 y 1 representativo del comportamiento observado.

2. El *remote tracker* recibe la notificación y mediante consulta a su propia caché o al servicio de metadatos observa que no le corresponde responder a esta notificación, por lo que redirige al cliente a su *home tracker* con la información almacenada en los pertinentes metadatos. La respuesta del *remote tracker* en este caso será idéntica a la efectuada en una consulta de reputación.

3. El cliente dirige de nuevo su consulta a su propio *home tracker*, gracias a la redirección `HTTP` obtenida del *remote tracker*.

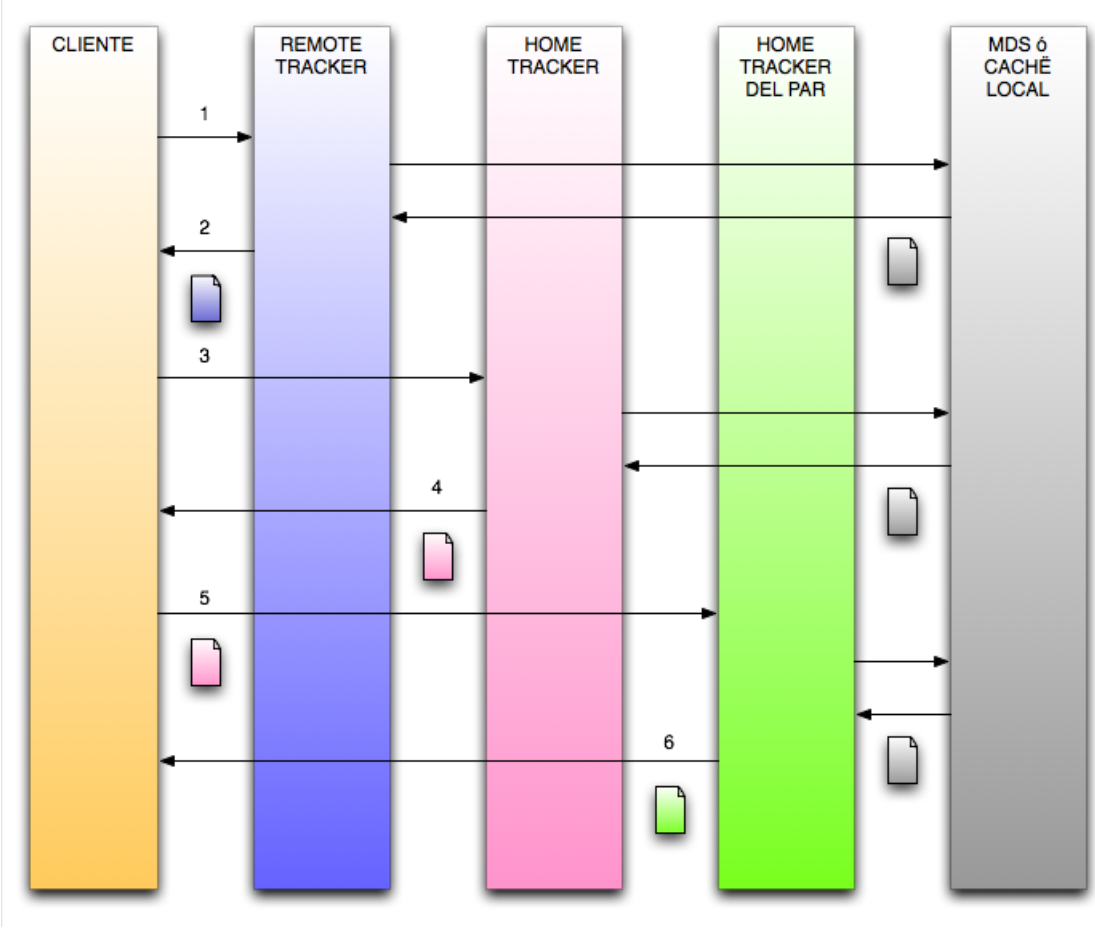

Figura 4.5: Diagrama de una notificación de reputación.

4. En este caso particular la respuesta del *home tracker* del cliente será una redirección `HTTP` al *home tracker* del par una vez determinada su ubicación (recordemos, bien mediante caché local, bien mediante consulta al servicio de metadatos), acompañada además de dos respuestas de autenticación de *eduGAIN*. Una certificará la identidad del propio cliente y añadirá un atributo con la reputación del mismo, de forma que el *home tracker* del par pueda utilizar dicha información para tomar una decisión en base a ella. Otra será relativa al propio *home tracker* del par, incluyendo su cadena completa de certificación, del mismo modo que ocurre en una petición de reputación.

5. Por último, el cliente formulará su petición al *home tracker* del par mediante un método `POST` de `HTTP`, incluyendo en el cuerpo de la consulta la respuesta de autenticación recibida de su *home tracker* relativa a sí mismo.

6. Gracias a esta respuesta de autenticación, el *home tracker* del par podrá verificar la identidad del cliente y decidir si ignora o no la notificación de reputación en base a un conjunto de criterios:

   - Sus propias políticas internas, que pueden prohibir explícitamente interacciones con clientes de otras organizaciones.
   - La reputación del propio cliente que realiza la notificación.
   - La ausencia de un identificador de transacción conocido que haya sido registrado previamente por una consulta de reputación, efectuada por el par sobre el cual se realiza la notificación en el momento actual.

   La respuesta del *home tracker* del par será, en cualquier caso, una respuesta `HTTP` con código de error 200 (*OK*) cuyo contenido es una respuesta de autorización de *eduGAIN* con resultado `Accepted` en caso de que se haya tenido en cuenta la notificación o `Deny` en caso contrario.

La figura 4.5 muestra el flujo de mensajes a las que puede dar lugar una notificación de reputación, así como aquellas en las que se adjunta un documento *SAML* como parte de la petición o la respuesta.

4.3.5. Protocolo de consulta de metadatos

El protocolo de consulta de metadatos tiene como objetivo permitir a los *trackers* del sistema obtener información relativa a otras organizaciones y de esta forma determinar a qué otros *trackers* debe dirigir, cuando proceda, al cliente que solicita la reputación de un par o que pretende realizar una notificación. De este modo, además, es posible establecer confianza entre otros *trackers* de la red y los propios clientes.

Pese a que en líneas generales el servicio de metadatos y su protocolo adjunto se rige por la definición del mismo que propone *eduGAIN*[edugain-arch], es necesario hacer una salvedad a dicha definición. En este caso, el servicio de metadatos supone el punto de entrada a la red por parte de las nuevas organizaciones que se adhieren a ella, por lo tanto es necesario establecer dos puntos de confianza en el servicio:

- En un primer lugar, la publicación de metadatos en el servicio se hará bajo estricto control de acceso garantizado única y exclusivamente a las entidades participantes de la red que, además, sólo podrán actualizar la información relativa a sí mismas.

- Por otro lado, a modo de garantía que permita asegurar que los resultados de una búsqueda no han sido alterados entre ambos extremos del canal de comunicaciones, se utilizará de forma obligatoria un canal asegurado mediante *SSL*[8], si bien no debe considerarse esto en ningún caso como forma de ocultar los metadatos, ya que estos son completamente públicos, y las organizaciones participantes deben tener esto en cuenta a la hora de decidir la información publicada en los mismos.

  Esta medida es necesaria ya que, tal y como se define en la especificación de *eduGAIN*, el servicio de metadatos es un simple almacén y en ningún caso dispone de lógica ni capacidad que le permita firmar los documentos en él publicados, por lo que se hace necesaria una forma alternativa de garantizar la integridad de los mismos. De este modo es posible establecer un grado de confianza en los metadatos de forma que se pueda asumir como fiable la información criptográfica que incluyen. En caso contrario, y al estar permitido que cada organización utilice su propia Infraestructura de Clave Pública mediante la publicación de la información de dicha infraestructura en el servicio de metadatos, un hipotético atacante podría obtener beneficios del sistema utilizando su propia *PKI* y la técnica de la intervención de las comunicaciones o *man in the middle*[9].

Por lo demás, el despliegue del servicio de metadatos y su protocolo asociado es exactamente idéntico al que definen las citadas especificaciones de *eduGAIN*. En este caso tan sólo es necesario realizar búsquedas de metadatos o *metadata searches*, utilizadas para realizar búsquedas atendiendo a diversos criterios, de los cuales la búsqueda por *URN* es la más adecuada. Por tanto, un *tracker* del sistema que desee obtener información acerca de una organización por la cual pregunta un par deberá realizar una búsqueda de metadatos incluyendo el *URN* recibido -el identificador de la entidad- en la lista de «localizadores de origen» o *home locators* que forma parte obligatoria de toda búsqueda. El servicio de

[8] *Secure Socket Layer*, «Capa de Sockets Seguros»[tls-rfc]

[9] La técnica del *man in the middle* es un conocido tipo de ataque que permite al atacante interceptar una conexión entre dos extremos haciéndose pasar por cada uno de ellos de cara al opuesto, consiguiendo de este modo acceso a toda la información intercambiada y pudiendo, por tanto, modificarla a su antojo.

metadatos responderá, en caso de encontrar la información solicitada, con un documento *XML* que sigue el esquema definido por el estándar *SAML* versión 2[saml-spec] y que contiene un elemento de tipo `EntityDescriptor` con toda la información proporcionada por la organización que lo publicó.

Dado que este es un proceso costoso, que implica la consulta al servicio de metadatos por cada organización participante de la red que no sea la misma a la que pertenece un determinado *tracker*, es posible realizar al menos un par de optimizaciones que reduzcan los recursos consumidos:

- Por un lado, un *tracker* con una carga suficientemente alta de consultas puede decidir crear una cola con dichas consultas de forma que se realice una sola búsqueda de metadatos para obtener los resultados que permitan responder a todas ellas. Para ello, el *tracker* incluirá cada *URN* identificativo de las organizaciones desconocidas en la lista de «localizadores de origen», de modo que la respuesta del servicio de metadatos será en esta ocasión un documento *SAML* versión 2 con un elemento `EntitiesDescriptor` dentro del cual se incluyen todos los resultados obtenidos en forma de elementos `EntityDescriptor` individuales.

- Adicionalmente, es posible almacenar los resultados de forma temporal en una memoria caché situada en el *tracker*, durante un margen razonable de tiempo, que permita evitar todas las consultas posibles al servicio de metadatos siempre y cuando ya se haya efectuado previamente una búsqueda de la organización a la que el par ha hecho referencia en su consulta.

4.4. Implementación

La implementación básica que se ha realizado a modo de demostración de la arquitectura aquí descrita está basada, como ya se ha mencionado previamente, en el cliente de *BitTorrent Vuze*, también conocido por el nombre *Azureus*. Gracias al esmerado diseño del mismo, ha sido posible integrar los protocolos y mensajes que forman parte de «aretusa» en forma de extensión o *plugin* del propio cliente, por lo que no ha sido necesario modificar ni una sola línea de código del mismo.

El desarrollo de la implementación se ha basado en el entorno integrado de desarrollo

(*IDE*) *NetBeans*[netbeans], lo que facilita la integración con los proyectos *Vuze* y *eduGAIN*, además de proporcionar una plataforma integrada de pruebas. El código fuente se mantiene bajo control de versiones mediante *Subversion*[subversion], en un repositorio creado exclusivamente para este proyecto en la forja de RedIRIS[forja]. Por último, se ha utilizado la herramienta *Ant* de *Apache*[ant] para la realización de pruebas unitarias en entornos *Windows*, *GNU/Linux* y *OSX*.

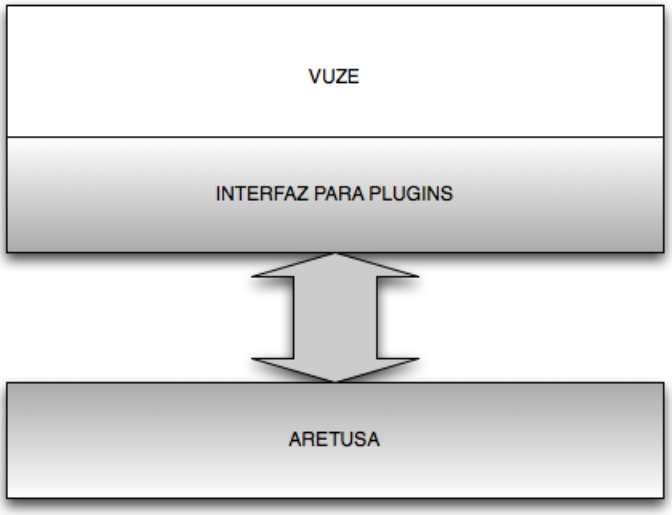

Figura 4.6: Diagrama de comunicación entre «Aretusa» y el interfaz ofrecido por *Vuze*.

El diagrama de la figura 4.6 describe el modelo de intercomunicación entre el *plugin* implementado y el cliente *Vuze* a través del interfaz de programación que proporciona para extensiones.

Para la correcta instalación y funcionamiento de las extensiones, *Vuze* impone una serie de sencillos requisitos que deben seguir todos los *plugins*:

- Las extensiones deben localizarse en un subdirectorio con el nombre de las mismas dentro del directorio `plugins` de la instalación del programa. Por ejemplo, nuestra implementación debe alojarse en el directorio `plugins/aretusa/`.

- Las extensiones se almacenan en el formato *jar* de Java, cuya raíz debe contener un fichero de propiedades de Java llamado *plugin.properties*. Dicho fichero debe contener, al menos, la siguiente propiedad:

plugin.class el nombre completo (incluyendo el paquete en el que se encuentra) de la clase Java principal que implementa la extensión, por ejemplo, en nuestro caso, `com.aelitis.azureus.plugins.aretusa.AretusaPlugin`.

De forma opcional, puede contener las siguientes propiedades:

plugin.langfile el nombre completo (incluyendo el paquete en el que se encuentra) de un nuevo fichero de propiedades de Java con las cadenas de texto originales de la extensión, por ejemplo, `com.aelitis.azureus.plugins.aretusa.ui.internat.Messages`. Utilizando este fichero como base, se pueden añadir tantos ficheros como idiomas se permitan utilizando códigos de lenguaje en el nombre de los ficheros. Por ejemplo, para la traducción al español de todos los mensajes incluidos en la extensión, utilizaremos el archivo `com/aelitis/azureus/plugins/aretusa/ui/internat/Messages_es_ES.properties`.

plugin.name un nombre sencillo para la extensión que será mostrada en la configuración de extensiones de *Vuze*.

plugin.id un identificador único para la extensión, utilizado por el proceso de actualización automática.

plugin.version el número de versión de la extensión, utilizado también para la actualización automática.

- La clase principal de la extensión debe implementar cualquiera de las dos clases `org.gudy.azureus2.plugins.Plugin` u `org.gudy.azureus2.plugins.UnloadablePlugin` (en caso de que deseemos permitir que la extensión pueda dejar de utilizarse en cualquier momento, sin necesidad de reiniciar *Vuze*.

Por otro lado, tal y como se ha venido mencionando a lo largo de este capítulo, nuestra implementación hace uso de la Infraestructura de Autenticación y Autorización *eduGAIN*, por lo que es necesario analizar también el interfaz ofrecido por la librería así como la forma de comunicación. En este caso, tanto *Vuze* como *eduGAIN* están escritos en el lenguaje Java, por lo que es posible utilizar directamente el interfaz que ofrece *eduGAIN* de forma nativa. Dicho interfaz se compone de tres librerías diferenciadas en tres paquetes:

net.geant.edugain.base contiene el conjunto básico de mensajes utilizados por *eduGAIN* así como los interfaces de los perfiles incluídos por defecto y las clases de configuración y definición de constantes utilizadas por la librería.

net.geant.edugain.validation proporciona servicios avanzados de firma digital de documentos y validación de las mismas, así como verificación de certificados y una interfaz altamente extensible para el manejo de identificadores de componente.

net.geant.edugain.meta proporciona servicios de consulta y publicación en el servicio de metadatos o *MDS*.

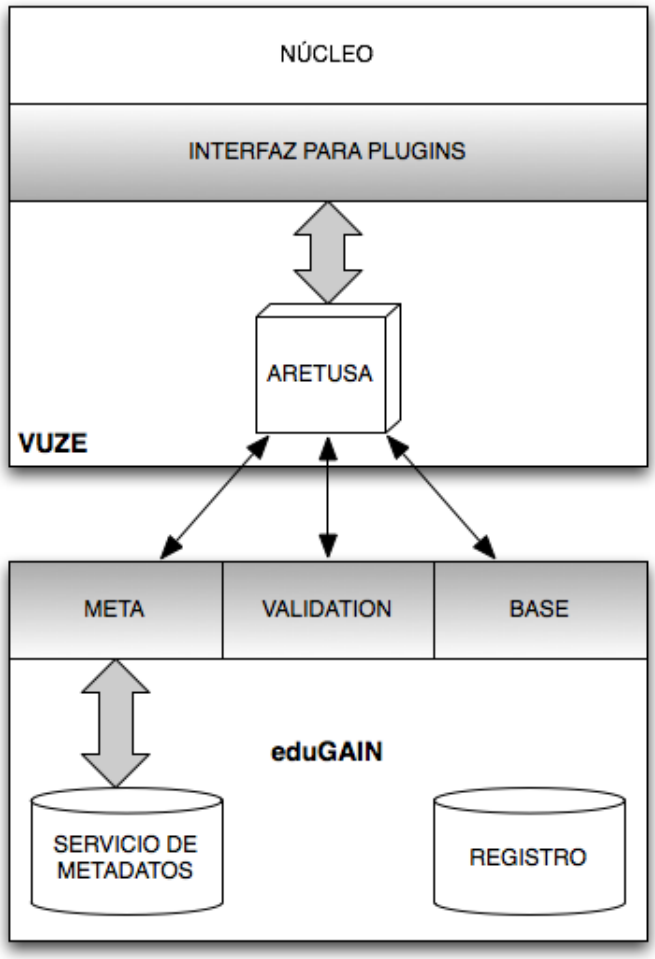

Figura 4.7: Diagrama de comunicación entre *Vuze*, la extensión «Aretusa» y *eduGAIN*.

Una vez cubiertos los requisitos necesarios y descrita la interoperabilidad de nuestra extensión con las librerías de las que hará uso, podemos comenzar con el diseño de las clases que componen «aretusa». En primer lugar es necesario definir las relaciones de la clase principal, `AretusaPlugin`, respecto del interfaz proporcionado por *Vuze*. Como ya se ha indicado, esta clase debe implementar obligatoriamente el interfaz proporcionado por `org.gudy.azureus2.plugins.Plugin`. Adicionalmente, tal y como se muestra en la figura 4.8, por conveniencia implementará a su vez los interfaces proporcionados por `org.gudy.azureus2.plugins.messaging.MessageManagerListener` y `org.gudy.azureus2.plugins.config.ConfigParameterListener`. De esta forma, resulta considerablemente más sencillo que la extensión se configure automáticamente durante su inicio, así como que gestione los conjuntos de pares a los que se encuentra conectado el cliente que soportan esta misma extensión.

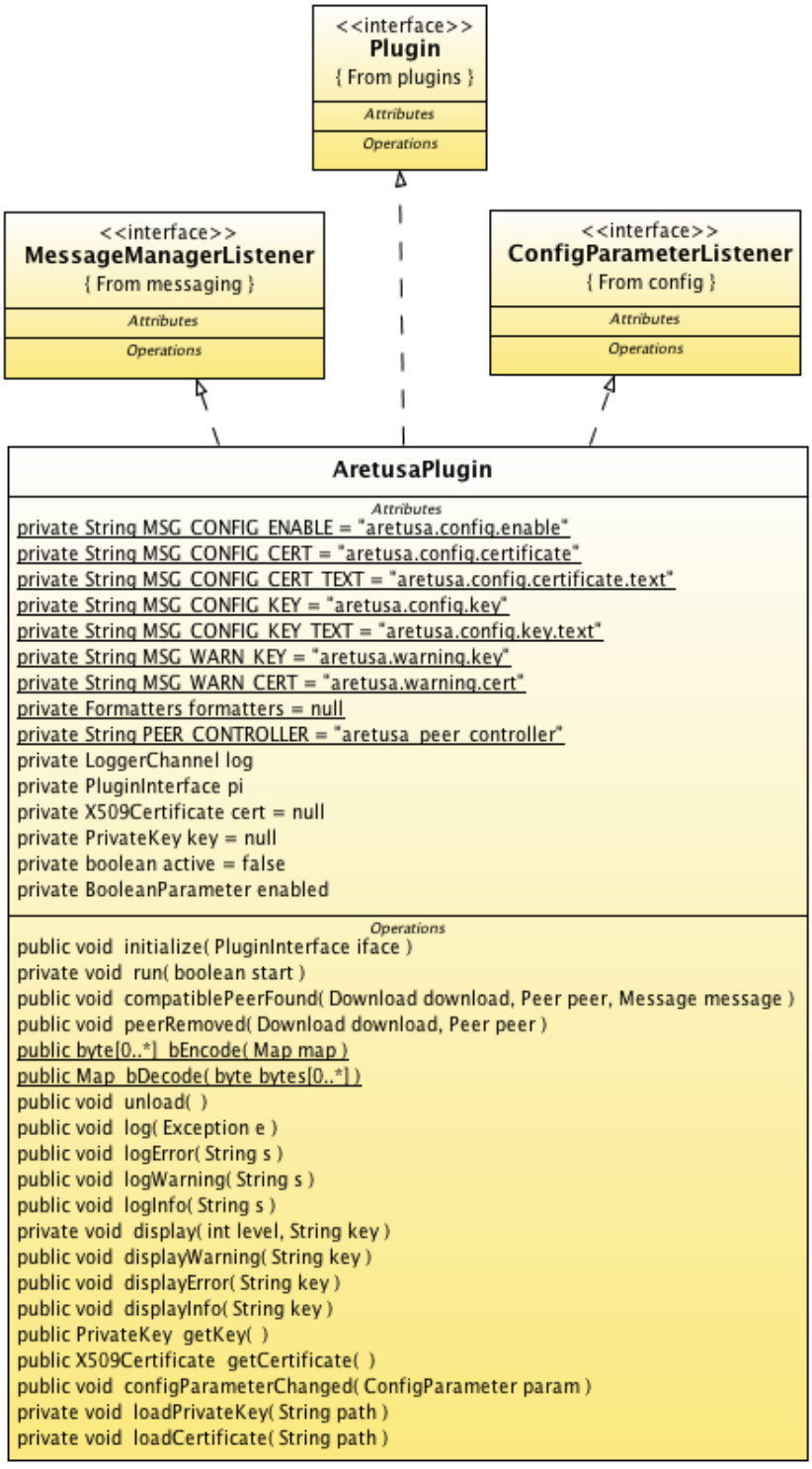

Figura 4.8: Diagrama de clases correspondiente a la clase principal de la extensión y su relación con el interfaz ofrecido por *Vuze*.

En la figura 4.9 se representan las distintas clases que componen la extensión y las relaciones de agregación existentes entre ellas. Por un lado, la clase que implementa la lógica de un *tracker* «aretusa», `ReputationHandler`, así como la clase que gestiona los pares compatibles con la extensión y su protocolo asociado, `PeerController`, hacen uso de la clase principal de la extensión para acceder al interfaz de programación de *Vuze*, así como para el acceso a métodos que permiten registrar toda la actividad generada.

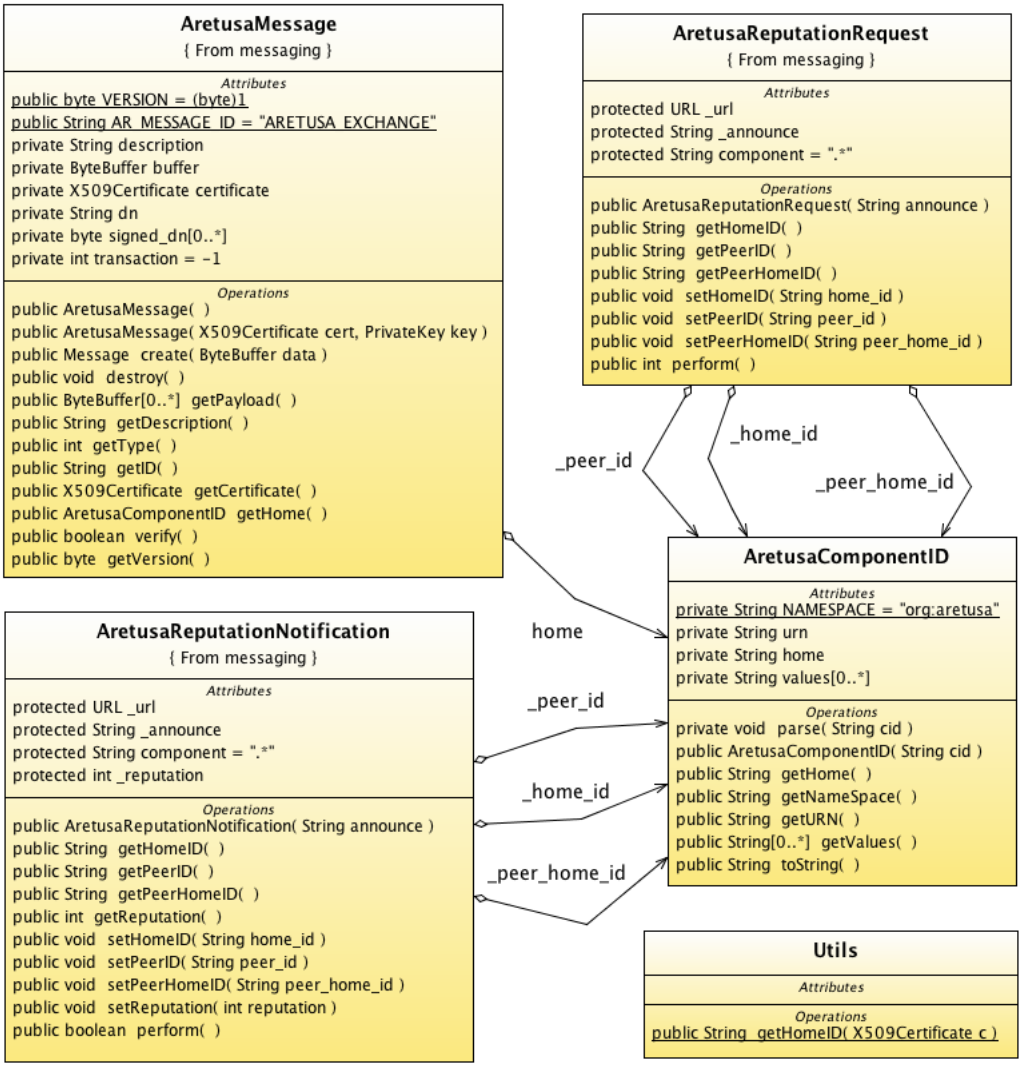

Figura 4.9: Primera parte del diagrama de clases correspondiente a todas las clases de la extensión «aretusa».

Por otra parte, en la figura 4.10 las clases que implementan el mensaje de intercambio de certificados, el protocolo de petición de reputación y el protocolo de notificación de repu-

tación, `AretusaMessage`, `AretusaReputationRequest` y `AretusaReputationNotification`, respectivamente, hacen uso de la clase que implementa los Identificadores de Componente de «aretusa» (`AretusaComponentID`) para validar los certificados obtenidos.

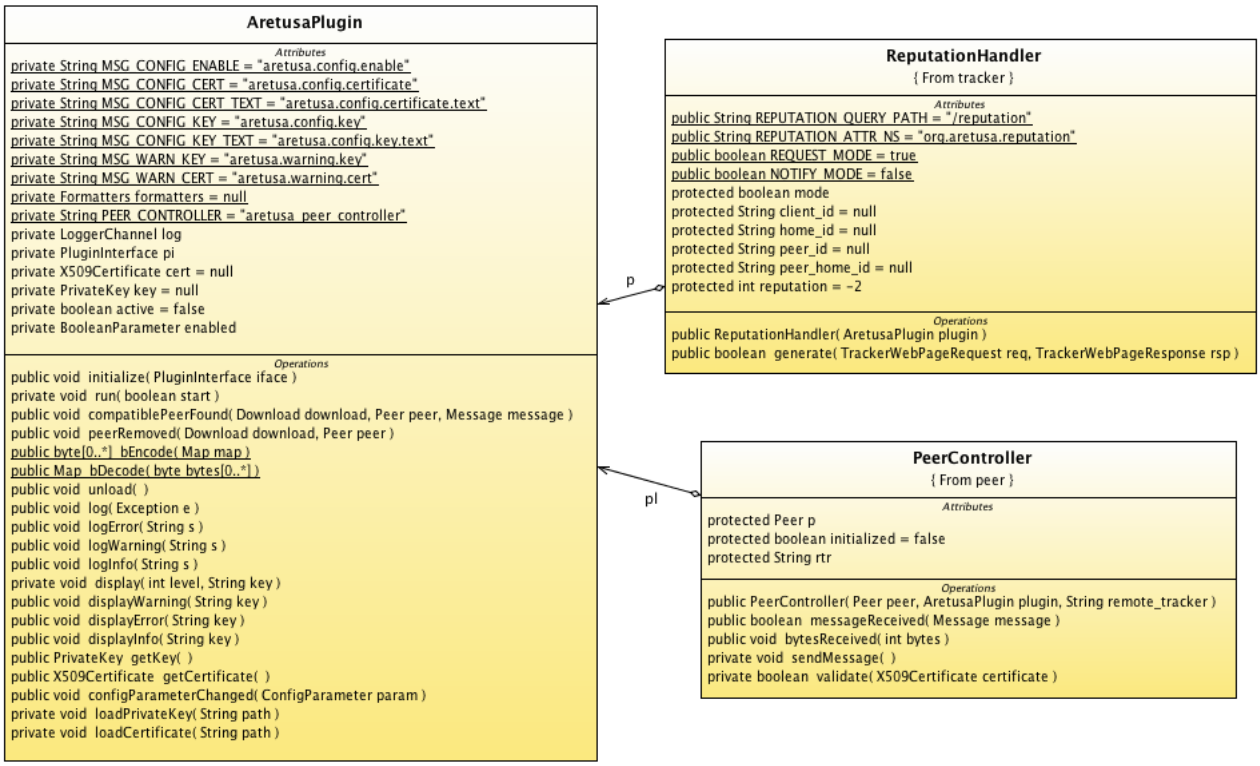

Figura 4.10: Segunda parte del diagrama de clases correspondiente a todas las clases de la extensión «aretusa».

La clase `PeerController` implementa el interfaz ofrecido por `org.gudy.azureus2.plugins.network.IncomingMessageQueueListener` y `org.gudy.azureus2.plugins.network.OutgoingMessageQueueListener` tal como se muestra en la figura 4.11, de forma que tiene acceso completo a las colas de mensajes recibidos y enviados. Esto permite gestionar de forma cómoda el tratamiento de los mensajes de intercambio de certificados, así como

el inicio de los protocolos de consulta y notificación de reputación, implementados por las clases `AretusaReputationRequest` y `AretusaReputationNotification` respectivamente.

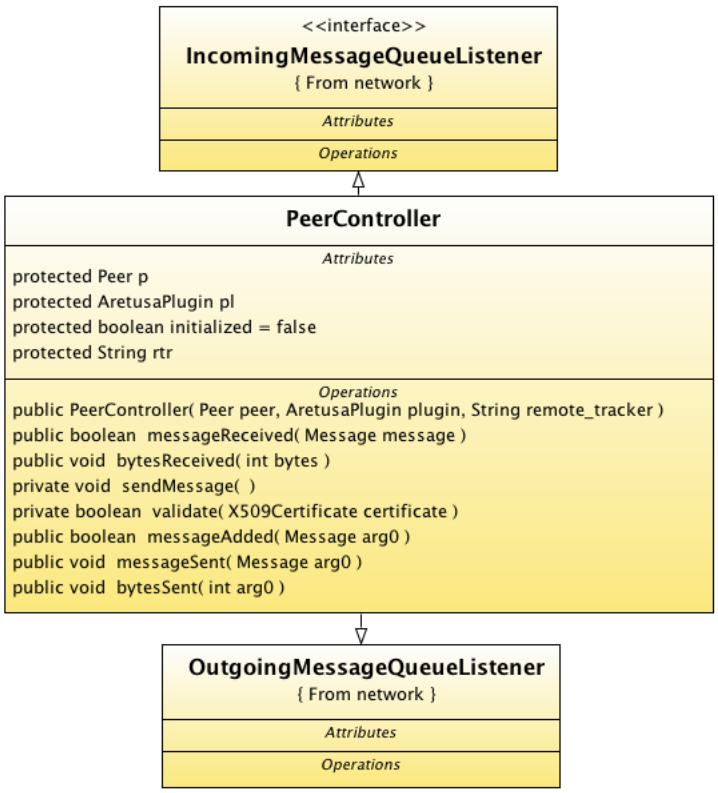

Figura 4.11: Diagrama de clases correspondiente al controlador de pares.

`AretusaMessage` es la clase que proporciona el tipo de mensajes de intercambio de certificados entre pares. Como se indica en la figura 4.12, implementa el interfaz `org.gudy.azureus2.plugins.messaging.Message` proporcionado por *Vuze* para todos sus mensajes a través del Protocolo de Mensajería de Azureus (*AZMP*). Al igual que ésta, tal y como se muestra en la figura 4.13, la clase correspondiente a los Identificadores de Componente de «aretusa», `AretusaComponentID`, implementa el interfaz `net.geant.edugain.validation.ComponentID`, en este caso proporcionado por *eduGAIN*, que define las características mínimas comunes a todos los Identificadores de Componente utilizables en el proceso de validación.

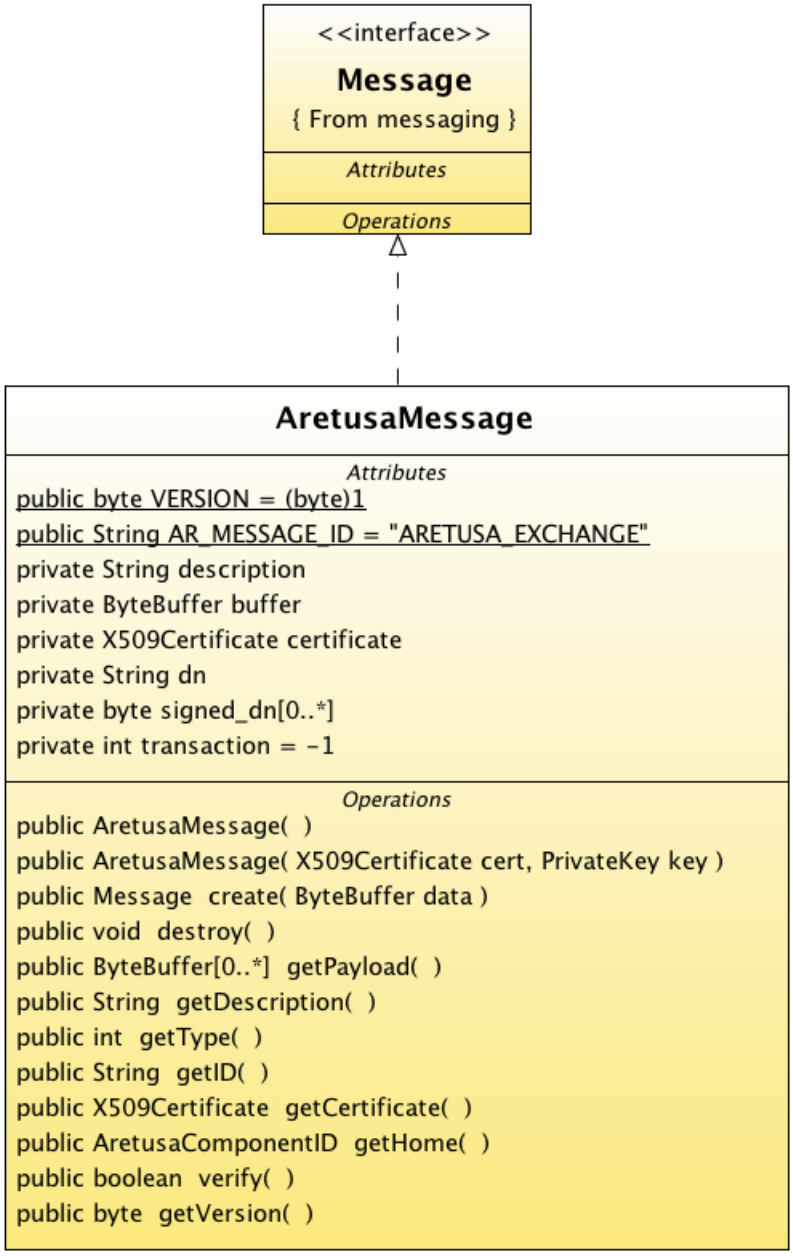

Figura 4.12: Diagrama de clases correspondiente al mensaje de autenticación de «aretusa».

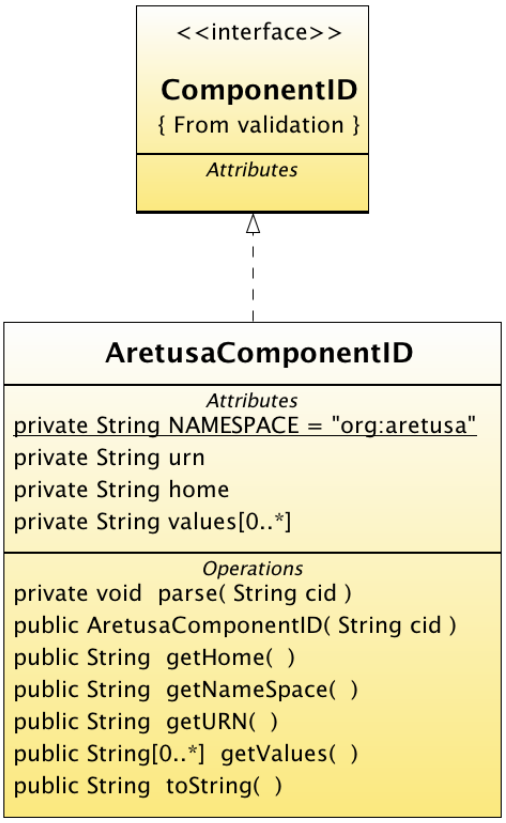

Figura 4.13: Diagrama de clases correspondiente a los identificadores de componente de «aretusa».

Por último, la clase `ReputationHandler` es responsable la lógica del *tracker* que permite recibir y responder de forma adecuada a cualquier petición o notificación de reputación. Para ello es necesario que implemente el interfaz `org.gudy.azureus2.plugins.tracker.web.TrackerWebPageGenerator`, encargado de servir cualquier página almacenada en el *tracker*, atendiendo al nombre y ruta completa de la misma. Este esquema de funcionamiento tiene una ventaja adicional, consistente en la facilidad a la hora de aplicar la convención de *scrapping*[10] definida por *BitTorrent* en cualquiera página que sirva el *tracker*.

[10]La convención de *scrapping* de *BitTorrent* determina el conjunto de peticiones `HTTP` válidas y reconocidas por el *tracker* a la hora de realizar acciones u obtener información acerca de un «torrent». Se puede consultar dicha convención así como algunas extensiones no oficiales a la misma en la especificación no oficial del protocolo[bt-spec-nonofficial].

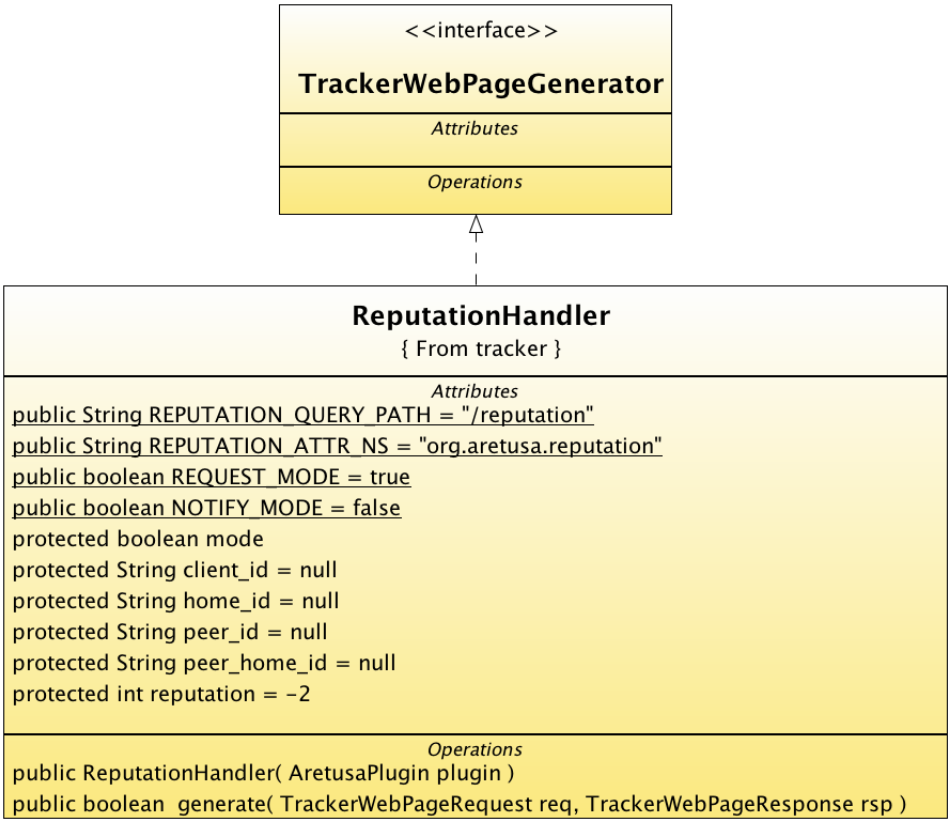

Figura 4.14: Diagrama de clases correspondiente al *tracker* de «aretusa».

Partiendo de este diseño, el funcionamiento de la extensión es sencillo. En primer lugar, *Vuze* llamará al método `initialize` de la clase principal cada vez que se cargue la extensión. Por tanto es este método el que se encargará de configurar toda la lógica del *plugin* para su puesta en funcionamiento inmediata. En concreto:

1. Creará la sección de configuración correspondiente a la extensión en el interfaz de usuario de *Vuze*, e iniciará los parámetros adecuados con configuraciones previamente guardadas, en caso de existir:

   certificado de usuario el certificado obtenido por el usuario de su organización para su uso en el sistema.

   clave privada la clave privada asociada a la pública referenciada en el certificado.

En caso de disponer de una configuración adecuada, «aretusa» podrá entrar en funcionamiento o ser desactivado en cualquier momento por parte del usuario.

2. Registrará en *Vuze* los mensajes de intercambio de certificados entre pares, de forma que desde ese mismo momento el cliente anunciará el soporte de la extensión «aretusa» y será capaz de gestionar la conexión con cualquier par que también la soporte.

3. Por último registrará la clase que gestiona las peticiones y notificaciones de reputación en el *tracker*, de forma que si el usuario habilita la utilización del mismo, éste ya se encontrará configurado y listo para su uso.

Adicionalmente, y a consecuencia de los interfaces que implementa la clase principal de la extensión, será la encargada tras el arranque inicial de:

- Reflejar cualquier cambio en la sección de configuración de «aretusa» en la configuración interna de la extensión, e incluso desactivarla en caso de que así se requiera.

- Actuar cada vez que se encuentre (o se desconecte) un par que soporta la extensión. Esta clase será notificada por parte de *Vuze* cada vez que tras la negociación inicial del Protocolo de Mensajería de Azureus se encuentre otro cliente compatible, al igual que cuando una conexión previamente establecida se interrumpa. De este modo, «aretusa» asociará un controlador (`PeerController`) a cada par encontrado que soporte el protocolo de intercambio de certificados entre pares.

De este modo, la lógica funcional del protocolo básico entre pares se encuentra alojada en la clase `PeerController`. La recepción de un mensaje procedente del par con su certificado y firma llevará asociada el envío de un mensaje equivalente con la información del cliente, tras el cual se produce invariablemente una consulta de reputación mediante la clase `AretusaReputationRequest`. Si la consulta finaliza de forma inadecuada o el resultado es una reputación negativa o nula, se marcará al par como ignorado (recordemos, *snubbed* según la definición del protocolo). En caso contrario -la consulta finalizó adecuadamente y el resultado es positivo-, sencillamente se permitirán las interacciones con dicho par.

Por otro lado, se observa la llegada de mensajes de tipo *piece*, aquellos que contienen bloques de datos según la definición de *BitTorrent*, lo cual provocará una notificación de reputación por parte del cliente, en este caso positiva. Además resulta necesario correlar los

mensajes *have* (en los que el par indica los «pedazos» del contenido original de los que ya se encuentra en posesión) con los mensajes de tipo *request* enviados desde el cliente. De este modo, si el par no responde adecuadamente a una petición tras haber admitido disponer del «pedazo» solicitado tras un determinado umbral de tiempo, el cliente realizará una notificación de reputación acerca del par, en este caso negativa.

Por último, la clase `ReputationHandler` implementa la lógica de las peticiones y notificaciones de reputación desde el punto de vista del *tracker* de forma sencilla. Cada vez que el *tracker* recibe una petición, *Vuze* ejecuta el método `generate` de dicha clase, pasándole como parámetro la susodicha petición. En este momento se verificará que la página a la que se refiere dicha petición es la correspondiente con una consulta o notificación de reputación, en cuyo caso se procederá a obtener los parámetros asociados y responder adecuadamente, bien sea mediante una redirección al *tracker* adecuado, bien mediante un documento *SAML* con la información necesaria.

4.4.1. Instalación

La instalación del cliente y del *tracker* es idéntica al estar basada en ambos casos en el cliente *Vuze*. El proceso es sencillo, y debe obedecer a los siguientes pasos:

1. Instalación del cliente *Vuze* acorde con las instrucciones específicas proporcionadas para el sistema operativo elegido.

2. Instalación del *plugin* «aretusa» en el directorio adecuado del cliente, lo que incluye la copia de todas las librerías necesarias (como por ejemplo *eduGAIN* y *OpenSAML*) al mismo directorio.

3. Configuración de *eduGAIN* acorde con las instrucciones particulares dependiendo del sistema operativo del usuario. Como detalle a tener en cuenta, el almacén de claves de confianza o *truststore* utilizado por *eduGAIN* debe contener la cadena de certificados completa de la organización a la que pertenece el usuario.

Tras este proceso, basta con ejecutar el cliente *Vuze*, buscar nuevas extensiones, seleccionar y configurar «aretusa» (indicando la localización del certificado y clave privada del usuario) y por último habilitarlo. Llegados a este punto, la extensión se encuentra lista para ser utilizada.

Capítulo 5

Conclusiones y trabajo futuro

La evolución de este proyecto no sólo ha implicado el desarrollo en el sentido estricto del software, sino además un completo estudio de las tecnologías presentes en la actualidad y el análisis del problema aquí presentado en las diversas redes de pares existentes, así como la viabilidad de una posible solución para el mismo, asunto que no es precisamente trivial y que sigue siendo objeto de estudio a día de hoy por parte de campos como la teoría de juegos.

Es por ello que el análisis del problema y de la posible solución han supuesto una gran parte del tiempo y esfuerzo consumido, fruto del cual se ha obtenido, adicionalmente a la implementación, la publicación de sendos artículos en las *Jornadas Técnicas de RedIRIS* del 2007[aretusa-jt07] y en la *TERENA Networking Conference* [1] del 2008[aretusa-tnc08], que fueron presentados durante ambos congresos y que pueden consultarse a través de las respectivas páginas web de ambos.

De este modo, desde los inicios del proyecto en los que predominaba el esbozo de una solución para el problema tras el análisis de las tecnologías existentes y la elección de *BitTorrent* como objetivo principal, se ha escrito un borrador de la arquitectura propuesta que ha ido evolucionando con el tiempo y que ha contado con la inestimable ayuda y consejo de expertos en tecnologías *P2P* o de seguridad, hasta llegar a una propuesta robusta y consistente que se ha podido materializar en la implementación definitiva.

Entrando en el terreno de dicha implementación, se ha elegido una metodología ágil de

[1] «Conferencia sobre Redes de TERENA». TERENA es una asociación internacional de redes educativas e investigadoras, que ofrece a sus socios un marco en el que colaborar, innovar y compartir conocimiento para el desarrollo de tecnologías sobre Internet, infraestructuras y servicios orientados a la comunidad académica.

desarrollo de software puesto que era la que más se adecuaba a los requisitos de este proyecto en particular. Éstos son: objetivo claro y persistente, pero una arquitectura cambiante que debe amoldarse a las particularidades del protocolo existente y que a la vez depende del desarrollo de *software* de terceros; sin restricciones temporales, lo importante era analizar los conceptos propuestos y estudiar su viabilidad práctica con resultados rápidos y sencillos de obtener, así como la fácil adaptación a los cambios producidos en la documentación técnica que se ha mantenido desde un inicio.

Es importante recalcar que el desarrollo del proyecto, al hacer uso de una plataforma desarrollada por terceros, en una organización un tanto dispersa y difusa, ha obligado a depender sobremanera de la misma e incluso mantener todo el proceso en suspenso hasta que dicha plataforma ha marcado unos hitos de funcionalidad y estabilidad. Si bien, también es cierto que el trabajo del autor como uno de los principales desarrolladores de *eduGAIN* ha permitido agilizar lo que en otra situación habría podido paralizar el proyecto, así como obtener un flujo constante de nuevas funcionalidades requeridas, que han podido ser implementadas de forma sencilla y rápida en *eduGAIN*.

Se puede decir, por tanto, que el esfuerzo correspondiente a la implementación no se limita solamente a la extensión del protocolo *BitTorrent* sobre un cliente existente a modo de demostración de la arquitectura propuesta, sino también al desarrollo conjunto de *eduGAIN*, toda una arquitectura de autenticación y autorización contenida en el marco de *GÉANT2*[2] y que permite la interoperabilidad y comunicación de todo tipo de aplicaciones y necesidades de identidad digital, entre ellas la que nos ocupa como parte de este proyecto.

Por tanto, y a modo de resumen, a lo largo del desarrollo se han realizado de forma secuencial una serie de trabajos a fin de cumplir los objetivos iniciales:

1. Análisis del problema del *free-riding* en las redes de pares.

2. Estudio de las principales redes de pares y el grado de afección que sufren, así como la elección de una de ellas para la propuesta de una posible solución.

3. Estudio de la posible solución al problema, atendiendo a las características técnicas y peculiaridades de la red elegida en el caso que nos ocupa, *BitTorrent*.

4. Elaboración de un borrador que presente la arquitectura elegida.

[2]GÉANT2 es la red académica de gran ancho de banda que sirve a la comunidad académica europea.

5. Refinamiento de la solución en diversas iteraciones, mediante consulta con expertos en la materia, y presentación de la misma en reuniones y congresos internacionales como forma de discusión que permita la constante mejora de la misma.

6. Finalización de la implementación de los perfiles básicos de *eduGAIN*, así como de las librerías de firma y validación en las que reside el modelo de confianza utilizado.

7. Depuración de las funcionalidades básicas de *eduGAIN* mediante la publicación de la librería y la constante obtención de resultados fruto de las pruebas realizadas por el grupo *JRA5*[3] del proyecto *GÉANT2*.

8. Implementación de la arquitectura propuesta como extensión al protocolo *BitTorrent* en forma de *plugin* independiente para el cliente *Azureus*.

9. Pruebas de concepto basadas en la implementación realizada.

10. Formalización de todo el trabajo realizado en la presente memoria mediante el lenguaje de marcado LaTeX[latex].

La consecución de todo este trabajo ha permitido obtener como resultado final una demostración práctica básica de la arquitectura propuesta y, por ende, la resolución de nuestro *leitmotiv*, el problema del *free-riding*.

5.1. Trabajo futuro

Si bien es mucho lo que se ha avanzado, es cierto que aún se puede desarrollar mucho trabajo entorno al que se ha descrito a lo largo de estas páginas. En ese sentido, a continuación se presentan algunas de las posibles líneas de actuación que pueden seguir a la conclusión de este proyecto:

- En primer lugar, y quizás más importante, es necesario construir una completa infraestructura de pruebas que permita analizar y mejorar la implementación, así como obtener datos y mediciones prácticas que avalen esta propuesta. Para ello habrá que recurrir a la virtualización de servicios y hacer uso de todos los recursos de hardware

[3]Grupo de trabajo perteneciente a GÉANT2, dedicado a tecnologías de movilidad, autenticación y autorización.

y redes de comunicaciones de los que se pueda disponer, ya que la escala de este tipo de redes es muy amplia e implica la utilización de múltiples nodos que representen diferentes roles en el sistema.

- Para llevar a cabo tales pruebas y disponer de una infraestructura adecuada, probablemente sea una buena idea aprovechar el proyecto *PASITO* de RedIRIS, una «red para investigar en red», que permite a sus usuarios crear redes virtuales sobre la actual infraestructura académica española sobre las que puedan realizar sus pruebas e investigaciones sin ningún tipo de limitación.

- En base a las pruebas extensivas propuestas, es posible analizar y evaluar la calidad de la representacion elegida para la reputación de los pares en el sistema. Probablemente otro tipo de representaciones resulten más eficientes o permitan un funcionamiento más adecuado de la red. Quizás, incluso, algunas de las restricciones aquí propuestas sean innecesarias por lo leve de los efectos negativos que tiene no aplicarlas. Sin embargo, todo ese tipo de apreciaciones es necesario realizarlas en base a datos empíricos que no se pueden obtener sino con una serie de exhaustivas pruebas.

- Al igual que la representación de la reputación, es posible analizar la conveniencia o no de permitir memorias caché en los clientes y servidores, de forma que pese a que se reduzca la fiabilidad de los datos, se gane notablemente en rendimiento. Nuevamente, para tomar una decisión de este tipo es necesario basarse en pruebas sobre la implementación ya existente. En ese sentido, dentro del marco del *Task Force on European Middleware Coordination and Collaboration* (*TF-EMC2*) de TERENA[4], se ha creado un grupo de trabajo específico para la investigación y desarrollo relativos a sistemas de reputación y tecnologías de redes entre pares, del que el autor de este proyecto ha sido propuesto como *activity leader*, lo que permitirá aprovechar los conocimientos y esfuerzos de expertos a nivel europeo en este campo y por tanto obtener los mejores resultados posibles.

- Una funcionalidad interesante que se puede añadir a la arquitectura es la detección de *free-riders* incluso antes de que actúen. Esto es posible mediante la implementación

[4] *TF-EMC2* es una actividad de TERENA cuyo principal objetivo es proveer un foro de discusión y fomentar la colaboración en asuntos de *middleware*.

de clientes «sonda» que compartan información con otros pares de forma aleatoria, y posteriormente les soliciten que compartan esos mismos datos, de forma que el *free-rider* quede al descubierto al negarse en última instancia.

- De mismo modo, el concepto de clientes «sonda» puede servir para detectar *trackers* con un comportamiento inadecuado en la red, realizando sobre ellos notificaciones de reputación y evaluando los resultados obtenidos, de forma que una red de estos clientes (cuya implementación podría ser requisito para participar en el sistema) podría servir de detección y alerta de abusos antes incluso de que lleguen a producirse.

- Actualmente, *eduGAIN* es un requisito tanto para la implementación del cliente como del *tracker*. Dado que la configuración de *eduGAIN* no es trivial, resulta interesante eliminarlo como requisito de los clientes, implementando un sencillo intérprete de *SAML* que permita obtener datos de las respuestas recibidas tanto en peticiones como en consultas de reputación.

- Por último, resulta deseable estudiar el grado de interés que puede suscitar esta arquitectura para su posible implementación en organizaciones, instituciones y empresas, desde universidades a proveedores de servicios, probablemente englobado en el marco de la iniciativa *P4P*. Un posible acuerdo que promocione el uso de «aretusa» podría potenciar sobremanera la utilización legítima de las redes de pares como forma de optimizar los recursos disponibles.

Apéndice A

Especificación de los mensajes

A.1. Mensajes de autenticación de pares

Los mensajes de autenticación enviados entre sí por los pares se corresponden con mensajes *BitTorrent* estándar cuyo contenido es un diccionario en formato *bencode* con los siguientes campos:

cert el certificado `X.509` acorde con las políticas de confianza de *eduGAIN* del par.

signature la firma del *Subject Distinguished Name* contenido en el certificado del par.

transaction un identificador de la transacción en curso que permita verificar que una notificación de reputación se efectúa tras un contacto real entre dos pares.

A.2. Mensajes de consulta de reputación

Los mensajes de consulta de reputación son dirigidos desde los pares a los *trackers* del sistema, en forma de petición `HTTP GET` estándar a la `URI /reputation` de los mismos (nótese que esta `URI` puede cambiar en función de la convención de *scrapping* de los *trackers*). Dicha petición contendrá los siguientes parámetros:

home_id el identificador de la organización de origen del cliente.

peer_id el identificador del cual se consulta la reputación.

peer_home_id el identificador de la organización de origen del par sobre el cual se consulta la reputación.

transaction_id el identificador de la transacción para la cual se solicita la reputación del par.

A.3. Mensajes de respuesta de reputación

Los mensajes de respuesta de reputación consisten en respuestas `HTTP` con una *AuthenticationResponse* de *eduGAIN* en el cuerpo de la respuesta y un código de estado adecuado dependiendo del *tracker* que emite la respuesta y a quién va dirigida.

A.3.1. Respuesta de un *remote tracker*

En el caso de una respuesta de reputación emitida por un *remote tracker* el código de error `HTTP` seleccionado será 302 (*Found*) y contendrá la cabecera `Location` con la dirección del *home tracker* del cliente a la que éste debe dirigirse.

El cuerpo de la respuesta de autenticación de *eduGAIN* contendrá los siguientes parámetros:

result `RedirectUserTo`.

[**location**] en caso de que la organización de origen del cliente disponga de más de un *tracker*, se incluirá un `AttrStatement` con un atributo `location` con la dirección de reputación de cada *tracker* disponible, acorde con la información reflejada en el servicio de metadatos.

A.3.2. Respuesta del *home tracker*

En el caso de una respuesta de reputación emitida por el *home tracker* del cliente el código de error `HTTP` seleccionado será 302 (*Found*) y contendrá la cabecera `Location` con la dirección del *home tracker* del par a la que éste debe dirigirse.

El cuerpo de la respuesta de autenticación de *eduGAIN* contendrá los siguientes parámetros:

result `ConnectTo`.

subject el *subject* del `AuthnStatement` y del `AttrStatement` se corresponde con el identificador (*URN*) del *home tracker* del par.

chain el `AttrStatement` contendrá un atributo `chain` con cada certificado perteneciente a la cadena de confianza que certifica al par.

[**location**] en caso de que la organización de origen del cliente disponga de más de un *tracker*, se incluirá un `AttrStatement` con un atributo `location` con la dirección de reputación de cada *tracker* disponible, acorde con la información reflejada en el servicio de metadatos.

A.3.3. Respuesta del *home tracker* del par

En el caso de una respuesta de reputación emitida por el *home tracker* del par el código de error `HTTP` seleccionado será 200 (*OK*).

El cuerpo de la respuesta de autenticación de *eduGAIN* contendrá los siguientes parámetros:

result `Accepted`.

subject el *subject* del `AuthnStatement` y del `AttrStatement` se corresponde con el identificador (*URN*) del par.

reputation el `AttrStatement` contendrá un atributo llamado *reputation* con el valor de la reputación del par.

A.4. Mensajes de notificación de reputación

Los mensajes de notificación de reputación son dirigidos desde los pares a los *trackers* del sistema, en forma de petición `HTTP GET` o `POST` estándar a la `URI /reputation` de los mismos (nótese que esta `URI` puede cambiar en función de la convención de *scrapping* de los *trackers*). Dicha petición contendrá los siguientes parámetros de forma general:

home_id el identificador de la organización de origen del cliente.

peer_id el identificador del cual se consulta la reputación.

peer_home_id el identificador de la organización de origen del par sobre el cual se consulta la reputación.

transaction_id el identificador de la transacción para la cual se solicita la reputación del par.

reputation un valor numérico descriptivo del comportamiento observado sobre el par.

A.4.1. Notificación a un *remote tracker*

En el caso de la notificación efectuada a un *remote tracker*, ésta se efectúa de la forma descrita mediante el método `HTTP GET`.

A.4.2. Notificación al *home tracker*

En el caso de la notificación efectuada al *home tracker*, ésta se efectúa de la forma descrita mediante el método `HTTP GET`.

A.4.3. Notificación al *home tracker* del par

En el caso de la notificación efectuada al *home tracker* del par, ésta se efectúa de la forma descrita mediante el método `HTTP POST`, añadiendo en el cuerpo de la petición la respuesta de autenticación de *eduGAIN* recibida en una respuesta previa del *home tracker* del cliente y destinada a la autenticación del mismo.

A.5. Mensajes de respuesta de notificación de reputación

Los mensajes enviados en respuesta a una notificación de reputación varían dependiendo de la relación del *tracker* notificado con el cliente.

A.5.1. Respuesta de un *remote tracker*

En el caso de una respuesta de reputación emitida por un *remote tracker* el código de error `HTTP` seleccionado será 302 (*Found*) y contendrá la cabecera `Location` con la dirección del *home tracker* del cliente a la que éste debe dirigirse.

El cuerpo de la respuesta de autenticación de *eduGAIN* contendrá los siguientes parámetros:

result `RedirectUserTo`.

[**location**] en caso de que la organización de origen del cliente disponga de más de un *tracker*, se incluirá un `AttrStatement` con un atributo `location` con la dirección de reputación de cada *tracker* disponible, acorde con la información reflejada en el servicio de metadatos.

A.5.2. Respuesta del *home tracker*

En el caso de una respuesta de reputación emitida por el *home tracker* del cliente el código de error `HTTP` seleccionado será 302 (*Found*) y contendrá la cabecera `Location` con la dirección del *home tracker* del par a la que éste debe dirigirse.

El cuerpo de la respuesta contendrá dos respuestas de autenticación de *eduGAIN*. La primera de ellas contendrá los siguientes parámetros:

result `ConnectTo`.

subject el *subject* del `AuthnStatement` y del `AttrStatement` se corresponde con el identificador (*URN*) del *home tracker* del par.

chain el `AttrStatement` contendrá un atributo `chain` con cada certificado perteneciente a la cadena de confianza que certifica al par.

[**location**] en caso de que la organización de origen del cliente disponga de más de un *tracker*, se incluirá un `AttrStatement` con un atributo `location` con la dirección de reputación de cada *tracker* disponible, acorde con la información reflejada en el servicio de metadatos.

La segunda de ellas, destinada a servir como credenciales del cliente frente al *home tracker* del par, contendrá los siguientes parámetros:

result `Accepted`.

subject el *subject* del `AuthnStatement` y del `AttrStatement` se corresponde con el identificador (*URN*) del cliente.

reputation el `AttrStatement` contendrá un atributo llamado *reputation* con el valor de la reputación del cliente.

A.5.3. Respuesta del *home tracker* del par

En el caso de una respuesta de reputación emitida por el *home tracker* del par el código de error `HTTP` seleccionado será 200 (*OK*).

El cuerpo de la respuesta de autorización de *eduGAIN* contendrá los siguientes parámetros:

result `Accepted` o `Deny`, según convenga, en el estado de la respuesta. `Permit` o `Deny` en coherencia con lo anterior, como atributo `Decision` del `AuthorizationDecisionStatement` incluído.

resource `org.aretusa.reputation` como valor del atributo `Resource` incluído endel el `AuthorizationDecisionStatement`.

consumer el identificador (*URN*) del par en el campo `ConsumerID` de la respuesta *eduGAIN*.

A.6. Consultas de metadatos

Los mensajes de consulta de metadatos se corresponden con mensajes estándar de consulta de metadatos definidos por la especificación de eduGAIN[edugain-arch], y más concretamente, con búsquedas de metadatos cuyo «localizador de origen» o *home locator* -o más de uno, si se desea una búsqueda de diversas organizaciones- se corresponde con el identificador único de la organización sobre la cual se requiere información.

Bibliografía

[azmp-spec] «Azureus Messaging Protocol.» URL http://www.azureuswiki.com/index.php/Azureus_messaging_protocol.

[bcep-spec] «BitComet Extension Protocol.» URL http://wiki.theory.org/BitTorrentSpecification#BitComet_Extension_Protocol.

[bt-spec-nonofficial] «BitTorrent Protocol Specification.» URL http://wiki.theory.org/BitTorrentSpecification.

[forja] «Forja de Conocimiento Libre de la Comunidad RedIRIS.» URL http://forja.rediris.es/.

[ltep-spec] «LibTorrent Extension Protocol.» URL http://www.rasterbar.com/products/libtorrent/extension_protocol.html.

[ant] «The Apache Ant Project.» URL http://ant.apache.org.

[freenet] «The Free Network Project.» URL http://freenetproject.org.

[edugain] «GÉANT Authorisation Infrastructure for the research and education community.» URL http://www.edugain.org/.

[latex] «The LaTeX Project.» URL http://www.latex-project.org/.

[netbeans] «The NetBeans Integrated Development Eenvironment.» URL http://www.netbeans.org/.

[uninett] «The Norwegian Research Network.» URL http://www.uninett.no/.

[subversion] «The Subversion Version Control System.» URL http://subversion.tigris.org/.

[vuze] «Vuze.» URL http://www.vuze.com.

[pki-rfc] Adams, C., y S. Farrell. «Internet X.509 Public Key Infrastructure Certificate Management Protocols.», marzo 1999. URL http://www.ietf.org/rfc/rfc2510.txt.

[free-riding-gnutella] Adar, Eytan, y Bernardo A. Huberman. «Free riding on Gnutella.» URL http://www.firstmonday.dk/issues/issue5_10/adar/.

[uri-rfc] Berners-Lee, T., R. Fielding, y L. Masinter. «Uniform Resource Identifier (URI): Generic Syntax.», enero 2005. URL http://www.ietf.org/rfc/rfc3986.txt.

[url-rfc] Berners-Lee, T., L. Masinter, y M. McCahill. «Uniform Resource Locators (URL).», diciembre 1994. URL http://www.ietf.org/rfc/rfc1738.txt.

[xml-spec] Bray, Tim, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, y John Cowan. «Extensible Markup Language (XML) 1.1.», agosto 2006. URL http://www.w3.org/TR/2006/REC-xml11-20060816/.

[bt-spec] Cohen, Bram. «The BitTorrent Protocol Specification.» URL http://bittorrent.org/beps/bep_0003.html.

[enp-spec] Crooks, Allan. «Extension Negotiation Protocol.» URL http://www.azureuswiki.com/index.php/Extension_negotiation_protocol.

[tls-rfc] Dierks, T., y C. Allen. «The TLS Protocol.», enero 1999. URL http://www.ietf.org/rfc/rfc2246.txt.

[nat-rfc] Egevang, K., y P. Francis. «The IP Network Address Translator (NAT).», mayo 1994. URL http://www.ietf.org/rfc/rfc1631.txt.

[skype-p2p] Guha, Saikat, Neil Daswani, y Ravi Jain. «An Experimental Study of the Skype Peer-to-Peer VoIP System.», febrero 2006. URL http://www.guha.cc/saikat/pub/iptps06-skype/.

[commons] Hardin, Garrett. «The Tragedy of the Commons.» *Science*, 162, nº 3859, (1968), 1243 – 1248. URL http://www.sciencemag.org/cgi/content/full/162/3859/1243.

[edonkey-spec] Heckmann, Oliver, y Axel Bock. «The eDonkey 2000 Protocol.» *Inf. téc.*, Darmstadt University of Technology, 2002.

[x509-rfc] Housley, R., W. Ford, W. Polk, y D. Solo. «Internet X.509 Public Key Infrastructure Certificate and CRL Profile.», enero 1999. URL http://www.ietf.org/rfc/rfc2459.txt.

[opensaml] Internet2. «OpenSAML.» URL https://spaces.internet2.edu/display/OpenSAML/Home.

[shibboleth] —. «Shibboleth.» URL http://shibboleth.internet2.edu/.

[bt-freeriding] Locher, Thomas, Patrick Moor, Stefan Schmid, y Roger Wattenhofer. «Free Riding in BitTorrent is Cheap.» En *HotNets V*. 2006. URL http://dcg.ethz.ch/projects/bitthief/.

[dht-spec] Loewenstern, Andrew. «DHT Protocol.» URL http://www.bittorrent.org/beps/bep_0005.html.

[edugain-arch] López, Diego R., Rodrigo Castro, B. Kerver, Thomas Lenggenhager, Mikael Linden, Ingrid Melve, Miroslav Milinovic, Jaime Pérez, Juergen Rauschenbach, Manuela Stanica, Klaas Wierenga, Stefan Winter, H. Ziemek, y Maurizio Molina. «GÉANT2 Authorisation and Authentication Infrastructure.» *Inf. téc.*, GÉANT2, 2008.

[edugain-cookbook] López, Diego R., José Manuel Macías, Maurizio Molina, Jaime Pérez, Juergen Rauschenbach, Andreas Åkre Solberg, y Manuela Stanica. «Guidelines for connecting to the eduGAIN AA Infrastructure.» *Inf. téc.*, GÉANT2, 2008.

[joost] MacCarthaigh, Colm. «Network Architecture of Joost.» En *Séptimo Foro de Operadores de Red de Reino Unido*. 2007. URL `http://www.uknof.org.uk/uknof7/MacCarthaigh-Joost.pdf`.

[kademlia] Maymounkov, Petar, y David Mazières. «Kademlia: a Peer-to-peer Information System Based on the XOR Metric.» En *Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems*. 2002.

[handbook-cryptography] Menezes, Alfred J., Paul C. van Oorschot, y Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001, 5 ed$^{\underline{\text{ón}}}$.

[urn-rfc] Moats, R. «URN Syntax.», mayo 1997. URL `http://www.ietf.org/rfc/rfc2141.txt`.

[p2pharnessing] Oram, Andy (ed.) *Peer to Peer: Harnessing the Power of Disruptive Technologies*. OReilly, 2001, 1st ed$^{\underline{\text{ón}}}$.

[aretusa-jt07] Pérez, Jaime. «Aretusa: sistema de reputación para BitTorrent.» En *Jornadas Técnicas de RedIRIS*. 2007.

[aretusa-tnc08] —. «Aretusa: reputation system for BitTorrent.» En *Terena Networking Conference*. 2008.

[radius-rfc] Rigney, C., S. Willens, A. Rubens, y W. Simpson. «Remote Authentication Dial In User Service (RADIUS).», junio 2000. URL `http://www.ietf.org/rfc/rfc2865.txt`.

[saml-spec] OASIS. «SAML 2.0 Specifications.», marzo 2005. URL `http://saml.xml.org/saml-specifications`.

[gnutella-spec] Distributed Search Services, Clip2. «The Gnutella Protocol Specification.» URL `http://www.stanford.edu/class/cs244b/gnutella_protocol_0.4.pdf`.

[x500-rfc] Wahl, M. «A Summary of the X.500(96) User Schema for use with LDAPv3.», diciembre 1997. URL `http://www.ietf.org/rfc/rfc2256.txt`.

[ldap-rfc] Wahl, M., T. Howes, y S. Kille. «Lightweight Directory Access Protocol.», diciembre 1997. URL `http://www.ietf.org/rfc/rfc2251.txt`.