REDES DE COMUNICACIONES



INFORME DE PRÁCTICAS DEL PROTOCOLO NANOP2P

**Miembros**:

* MOURAD ABBOU AAZAZ
* PABLO JOSÉ ROCAMORA ZAMORA

**Grupo**: 2

**Profesor**: ÓSCAR CÁNOVAS REVERTE

**ÍNDICE**

**DESCRIPCIÓN**

Esta practica de Redes de Comunicaciones consiste en el diseño y la implementación de un sistema de transmisión de información entre pares (peers) y un servidor(tracker) que cuenta con una base de datos que contiene todos los metadatos e información de los peers que quieren compartir sus ficheros.

La realización de este proyecto la llamaremos NanoP2P, ya que simula al protocolo P2P pero de manera mas simple y reducida.

El mecanismo principal es que cuando un peer quiere compartir sus ficheros, debe darse alta en el tracker, y enviarle la lista de ficheros que contiene en su carpeta local. El tracker recibe información de sus ficheros y los almacena en su base datos.

Este proceso también es necesario para los peers que quieren descargarse ficheros de otros peers. Primero deben darse de alta en el servidor, para acceder a la información de los peers que también están dados de alta. Cuando un peer se descarga un fichero, debe notificar al tracker que lo va a incluir en su lista de ficheros locales.

Durante la descarga de ficheros entre peers, uno hará de servidor (seeder) y otro de cliente (downloader).

La realización de esta practica consta de dos partes:

* **Diseño** : Especificación de formatos de mensaje y autómatas que definan la pasos que debe seguir el protocolo.
* **Implementación**: Implementación en Java de la practica y de los formatos de la parte de diseño, siguiendo los estados del autómatas.

**DISEÑO**

En esta fase, se diseñan los nuevos formatos que usaremos para los mensajes que se utilizaran para la comunicación entre peers.

**FORMATOS DE MENSAJE**

Los campos que pueden aparecer en los tipos de mensaje son:

* **Type:** Indica el tipo de mensaje. Su tamaño es de 1 byte:

Dependiendo del numero que sea puede ser :

- 1: ***GET\_CHUNK*** :Un Peer solicita al otro Peer la lista de Chunks que tiene de un determinado fichero,

que indica a través de su Hash.

- 2: ***GET\_CHUNK\_RESPONSE***: Un Peer informa a otro Peer de la lista de chunks que tiene de un determinado fichero.

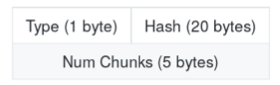
- 3 : ***CHUNK***: Un Peer solicita al otro Peer un Chunk de un fichero específico indicado a través de su Hash.

- 4 **: CHUNK\_RESPONSE:** Un Peer manda a otro Peer el chunk que le ha solicitado.

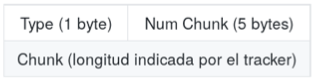
**- Hash :** Es de 20 bytes e indica el hash que identifica a un fichero.

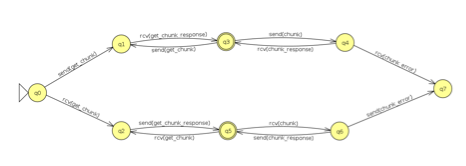
**-** **Num Chunk:** Número del chunk del que procede el dato. Es de 5 bytes.

**- Chunk :** Dato del chunk, el tamaño se establece acorde al tamaño que nos indica el Tracker.

FORMATO GET\_CHUNK Y CHUNK

FORMATO GET\_CHUNK\_RESPONSE Y CHUNKRESPONSE



**AUTÓMATAS DEL PROTOCOLO PEER-PEER**

**Estado** q0: Es el estado inicial. En este estado el peer, puede tiene dos opciones:

- Pasar al estado cliente (q1)

- Pasar al estado servidor (q2)

**Estado** q1: Este estado se da cuando un peer cliente manda una petición de get\_chunk al peer servidor. Si no hay errores, el peer cliente debe recibir una lista de chunks y pasa al estado q3.

**Estado** q2: Este estado se da cuando un peer servidor recibe una petición get\_chunk de un peer cliente. Si no hay errores, el peer servido debe enviar una lista de chunks y pasa al estado q5.

**Estado** q3: Este estado se da cuando un cliente ha recibido una lista lista de chunks de un fichero. A partir de ahí podrá solicitar trozos de fichero.

**Estado** q4: Este estado ocurre cuando se ha enviado una solicitud un trozo de fichero (chunk). En este estado, volverá al estado q3 si ha recibido el trozo de fichero, o irá al estado q7 en caso de que ocurra algun error.

**Estado** q5: Este estado se da cuando el peer servidor envía una lista de chunks de un fichero al peer cliente. Se mantiene en el estado hasta que se recibe una solicitud de trozo o se vuelve a pedir de nueva una lista de trozos disponibles

**Estado** q6: Este estado se da cuando el peer servidor recibe una solicitud de trozo de fichero. En este caso el peer servidor manda un trozo de fichero al cliente en el que vuelve al estado anterior, o bien notifica un error pasando al estado siguiente.

**Estado** q7:Estado de error que ocurre cuando ha habido algún fallo en el socket de transmisión. Esto puede deberse a que un peer se da de baja o hay un fallo de red.

**IMPLEMENTACIÓN DEL**

**PROTOCOLO NANOP2P**

La implementación del protocolo se hace en Java. Parte de un conjunto de paquetes predeterminados que se nos proporciona inicialmente, y de ahí debemos ir desarrollando cada una de las partes del proyecto para que el envío de ficheros sea satisfactorio.

El proyecto contiene las siguientes clases:

-  **es.um.redes.P2P.App**: Este paquete contiene las clases Peer.java y Tracker.java. También están contenidas las clases PeerController que es la clase coordinadora, PeerCommands y PeerShel para gestionar los comandos y sus argumentos.Aquí se ejecutan los métodos main que lanzan en ejecución el peer y el tracker.

* **es.um.redes.P2P.PeerPeer.Client**: Contiene todas las clases relacionadas con el Peer cliente: Downloader, DownloaderThread, Estado y ProgressBar. Estas dos últimas fueron creadas para el estado y el resumen de la descarga.

-  **es.um.redes.P2P.PeerPeer.Message**: Contiene la clase Message.java que representa el formato de mensaje definido en el diseño que se transmite entre peers para el protocolo peer-peer.

-  **es.um.redes.P2P.PeerPeer.Server**: Contiene las clases que representan la parte servidor del peer: Seeder.java y SeederThread.java.

* **es.um.redes.P2P.PeerTracker.Client**: Este paquete contiene las clase Reporter cuya función es establecer la comuniciación entre un peer y el tracker.

-  **es.um.redes.P2P.PeerTracker.Message**: Contiene las clases que modelan los mensajes del protocolo peer-tracker soportados por la implementación del tracker. Las clases son: Message.java, MessageControl.Java, MessageDataFileInfo.java, MessageDataSeedInfo.java.

* **es.um.redes.P2P.PeerTracker.Server**: Contiene las clases TrackerThread.java la cual implementa la funcionalidad del servidor en el protocolo peer-tracker y la clase TrackerDataBase.java que implementa la base de datos.
* -  **es.um.redes.P2P.PeerTracker.util**: Contiene las clases FileDigest.java, FileInfo.java, FileInfoPath.java y FileInfoSeeds.java que sirven para construir la base de datos del tracker. También contiene la clase Ficheros que la usaremos para la lectura y escritura de trozos para el peer-peer.

**CLASES DESARROLLADAS**

**CLASE Peer.java**

Esa clase representa un peer. Este peer contiene una base de datos donde almacena sus ficheros y toma como argumento, la dirección IP del tracker y la ruta del directorio de ficheros.

Esta clase dispone de un objeto **Reporter** y un **PeerController** que actúa como controlador con el tracker.

También dispone de un **Seeder** que actúa como servidor de ficheros en caso de que reciba solicitudes de trozos. El peer al iniciarse se da de alta con el tracker, informándole de los ficheros que ofrece.

En ese momento se mantiene al espera de recibir un comando o una orden hasta que se el usuario decida cerrar el peer con un comando *quit.*

La comunicación que realiza con el tracker es mediante el protocolo UDP, y TCP para la comunicación con los demás peers

**CLASE Reporter.java**

La clase Reporter es la clase por la que un Peer establece comunicación con el tracker. Para ello, la clase reporter dispone de los atributos **trackerHostname** para establecer conexión, un **DatagramSocket**, que es un socket UDP con el que se conecta al tracker y un **inetSocketAdress** que será la direccion del tracker. Esta clase es usada por el peer y el usuario no interactúa con ella. Su funcion principal es enviar y recibir mensajes de comunicación con el tracker.

Sus métodos principales son:

* ***sendMessageToTracker*:** Esta función recibe como parámetro un socket UDP, un mensaje y la direccion del tracker. Convierte el mensaje en un array de bytes y lo envía por el socket UDP.
* ***receiveMessageFromTracker*:** Esta función recibe como parámetro el socket UDP en la que se recibe un mensaje del tracker y lo procesa.
* ***conversationWithTracker*** : inicia la comunicación con el tracker, mediante la transmisión y el recibo de mensajes con los dos métodos anteriores mencionados.

**CLASE PeerController.java**