

MEMORIA REDES DE COMPUTADORES - Nanofiles
Ángel Ruiz Fernandez & Marco Aurelio Santos Gómez
2º de Informatica - Grupo 2.2 - Profesor Eduardo Salvador Iniesta Soto

TABLE OF CONTENTS

1. Introducción	3
2. Protocolos	3
2.1. Directorio	3
2.1.1. Mensajes de cliente	3
2.1.1.1. Ping request	3
2.1.1.2. Filelist request	3
2.1.1.3. Publish request	4
2.1.2. Mensajes del directorio	4
2.1.2.1. Ping reply	4
2.1.2.2. Ping bad reply	4
2.1.2.3. Filelist reply	4
2.1.2.4. Publish reply	5
2.2. Peer-to-Peer	5
2.2.1. Mensajes de cliente	5
2.2.1.1. File request	5
2.2.1.2. Chunk request	5
2.2.1.3. Stop download	5
2.2.2. Mensajes de servidor	6
2.2.2.1. Accepted	6
2.2.2.2. Bad file request error	6
2.2.2.3. Chunk	6
2.2.2.4. Bad chunk request error	7
3. Mejoras	7
4. Capturas de pantalla de Wireshark	7
5. Conclusiones	7
6. Valoración personal	7
6.1. Por qué el proyecto base de Nanofiles es malo: Un análisis	7
6.1.1. Elección del lenguaje	7
6.2. Problemas con el código	8
6.2.1. General	8
6.2.2. Especifico	8

1. Introducción

El presente documento tiene el objetivo de describir el desarrollo del proyecto de prácticas Nano-Files de la asignatura Redes de Comunicaciones. Cursada en el segundo año del Grado Ingeniería en Informática de la Universidad de Murcia.

El proyecto consiste en el desarrollo de un sistema de compartición y transferencia de archivos en red usando una metodología Peer-to-Peer. A continuación se especifican las decisiones de diseño tomadas.

2. Protocolos

El sistema está conformado por un conjunto de peers que pueden compartir archivos entre sí usando el protocolo TCP. En adición, los peers mantienen contacto con un servidor directorio que contiene información general del servicio Nano-files. Dice qué archivos están disponibles y qué peers sirven dichos archivos. La comunicación con dicho directorio se efectúa mediante protocolo UDP.

En esta sección se especifican los formatos de los mensajes usados en la comunicación Peer-Peer y Peer-Directorio. Así como el comportamiento de estos en cada momento del intercambio de información.

Notación

- asdf: static data
- < >: mandatory field
- []: optional field
- [...]: last object N times
- All sizes are in bytes

2.1. Directorio

Codificados en texto plano UTF-8 siguiendo el formato codigo:valor sin espacios en blanco y sobre protocolo UDP.

2.1.1. Mensajes de cliente

2.1.1.1. Ping request

Prueba la comunicación con el directorio y la compatibilidad del protocolo.

- Operación: 'ping'
- Campos: Identificador de protocolo.
- Respuesta: ping reply

```
operation: ping
protocol: <protocol id>
```

2.1.1.2. Filelist request

Solicita información de los archivos conocidos por el directorio.

- Operación: 'filelist'
- Campos: ninguno.
- Respuesta: filelist reply

```
operation: filelist
```

2.1.1.3. Publish request

Informa al directorio de la lista de archivos disponibles para descargar desde este cliente.

- Operación: `publish`
- Campos: (ver más abajo), la lista puede estar vacía.
- Respuesta: publish response

```
operation: publish
[port: <port>]
file: <hash1>; <filename1>; <size1>
file: <hash2>; <filename2>; <size2>
[...]
```

2.1.2. Mensajes del directorio

2.1.2.1. Ping reply

Respuesta de ping informando la compatibilidad del protocolo.

- Operación: `pingok`
- Campos: ninguno.
- Responde a: ping request

```
operation: pingok
```

2.1.2.2. Ping bad reply

Notifica al cliente de que usa el protocolo incorrecto.

- Operación: `pingbad`
- Campos: ninguno.
- Respuesta a: ping request

```
operation: pingbad
```

2.1.2.3. Filelist reply

Devuelve una lista con información de todos los ficheros conocidos por el directorio (nombre, firma hash, tamaño y peers)

- Operación: `filelistres`
- Campos: (ver más abajo), la lista puede estar vacía.
- Respuesta a: filelist request

```
operation: filelistres
file: <hash1>; <filename1>; <size1>; <server1a>, <server1b> [...]
file: <hash2>; <filename2>; <size2>; <server2a> <server2b> [...]
[...]
```

2.1.2.4. Publish reply

Confirmación del mensaje publish request.

- Operación: 'publishack'
- Campos: ninguno.
- Respuesta a: publish request

operation: publishack

2.2. Peer-to-Peer

Mensajes con formato binarios representados en little-endian sobre protocolo TCP.

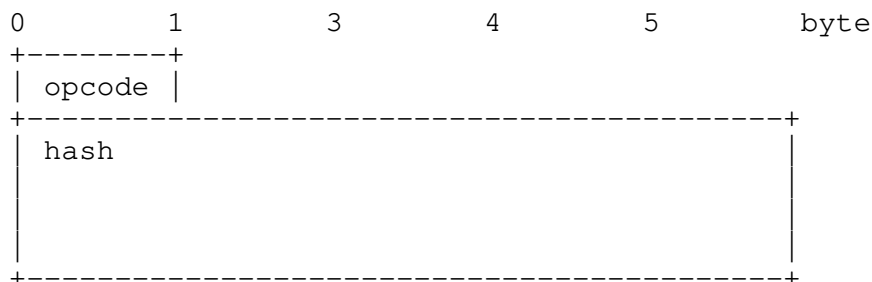
Todos los mensajes comienzan con un código de operación.

2.2.1. Mensajes de cliente

2.2.1.1. File request

Consulta la disponibilidad de un archivo para ser descargado.

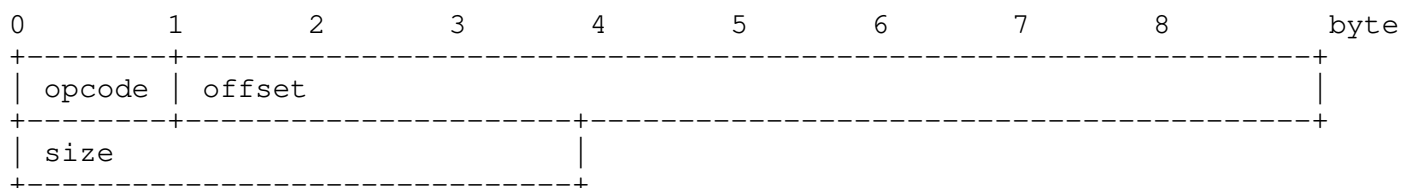
- Código: 0x01
- Campos:
- - Hash[20]: firma hash del fichero.
- Respuestas: accepted o file not found error



2.2.1.2. Chunk request

Solicita al servidor el envío de un archivo.

- Código: 0x02
- Campos:
- - Offset[8]: byte de inicio del chunk.
- - Size[4]: tamaño del chunk.
- Respuestas: chunk o bad chunk request error



2.2.1.3. Stop download

Termina la descarga actual de un fichero.

- Código: 0x03
- Campos: ninguno.
- Respuestas: ninguna.

```

0
+-----+
| opcode |
+-----+

```

2.2.2. Mensajes de servidor

2.2.2.1. Accepted

Informa de la disponibilidad de un fichero para ser descargado vía chunk requests.

- Código: 0x11
- Campos: ninguno.
- Respuesta a: file request

```

0
+-----+
| opcode |
+-----+

```

2.2.2.2. Bad file request error

Fichero no disponible o no encontrado.

- Código: 0x12
- Campos: ninguno.
- Respuesta a: file request

```

0
+-----+
| opcode |
+-----+

```

2.2.2.3. Chunk

Chunk de datos del fichero.

- Código: 0x13
- Campos:
 - size[4]: tamaño del chunk.
 - data[size]: datos.
- Respuesta a: chunk request

```

0          1          2          3          4          byte
+-----+-----+-----+-----+-----+
| opcode | size                                     |
+-----+-----+-----+-----+-----+
| data ...

```

2.2.2.4. Bad chunk request error

Fichero no disponible o no encontrado.

- Código: 0x14
- Campos: ninguno.
- Respuesta a: chunk request

```
0
+-----+
| opcode |
+-----+
```

3. Mejoras

4. Capturas de pantalla de Wireshark

5. Conclusiones

6. Valoracion personal

6.1. Por qué el proyecto base de Nanofiles es malo: Un analisis

6.1.1. Elección del lenguaje

Java era un lenguaje ampliamente usado para escribir software nuevo... hace 15 años. Ahora se considera como mala opción, como deuda tecnica (por razones fuera del alcance de este documento) no se escribe nuevo software en el, solo se mantiene el antiguo porque es mas barato que reescribirlo.

Específicamente, Java es una opción terrible para este caso porque es un obstaculo para el objetivo del proyecto: aprender a escribir aplicaciones con sockets.

Java, siendo OOP puro, está sujeto a limitaciones que le impiden representar fielmente realmente como funcionan las cosas. La API tiene una serie de clases para representar datagramas UDP por ejemplo: los datagramas existen, pero la clase DatagramPacket es algo muy abstracto que en ningún momento representa el paquete real, sino juntar datos en una unidad porque sí. Java no sabe lo que es un datagrama realmente.

Que queremos enviar un cacho de datos a varias direcciones: tenemos que construir un DatagramPacket para cada vez, con los mismos datos, pero con una dirección de destino distinta. Java está lleno de estas construcciones, copias copias y copias y destrucciones completamente innecesarias.

Cada día nos alejamos mas de C: La API de sockets de Java está demasiado alejada de la verdad: la API de sockets POSIX. Esa que todos los sistemas operativos reales implementan de una forma u otra desde BSD 4.2 (incluso Windows), que es lo que la JVM realmente usa por debajo para hablar con la red.

Esta API POSIX, tenemos unos datos, y los enviamos a una direccion, en una sola llamada, y sin ningún tipo de construcción o copia de nada. Es el kernel el que coje el búfer de memoria, construye el paquete UDP, lo encapsula en un frame Ethernet y se lo pasa al driver de la tarjeta de red (generalmente).

- Java no está diseñado para trabajar con datos binarios fácilmente, teniendo que serializar y deserializar todo, y ni si quiera soporta tipos enteros sin signo explícitamente.
- Aparentemente no hay forma de coger un trozo de un archivo y mandarlo por un socket sin copiar el buffer, lenguaje inutilizable, como se permite esto en 2025?
- Java es muy inflexible en cuanto a operaciones no bloqueantes. POSIX tiene la syscall poll() para monitorear una lista de sockets en un solo thread, y Java tiene algo parecido pero crucialmente, en C puedes responder con llamadas bloqueantes sabiendo que no se van a bloquear, mientras que en Java solo puedes usar cosas no bloqueantes sobre un canal con select, lo cual es pedantico y molesto.

6.2. Problemas con el código

6.2.1. General

- Legibilidad horrible
- Nombres confusos, ambiguos
- Inconsistencias de patrones
- Innecesariamente complicado
- Verbosidad (mas culpa de Java supongo)
- Exceso de comentarios, y además confusos
- Capa sobre capa sobre capa de abstracciones: paso de datos entre clases que no hacen nada innecesariamente
- Enumeraciones, arrays, estructuras innecesarias, como los comandos.

6.2.2. Especifico

Nos dan el proyecto para Eclipse, usando el build system de eclipse. Con lo cual nos obligan a usar Eclipse. Los IDEs son un obstaculo para el aprendizaje. Tienen una cantidad de ruido visual increíbles, son enormemente complicados de usar y suelen tener bugs y molestias. La alternativa es "convertirlo a gradle" y usar gradle que es otro build system terrible y innecesariamente complejo.

Ejemplo de TODO ambiguo: En cual se crea el hilo?

- NFControllerP2P: "Arrancar servidor en segundo plano creando un nuevo hilo"
- NFServer: "Añadir métodos a esta clase para: 1) Arrancar el servidor en un hilo nuevo que se ejecutará en segundo plano" La verdadera respuesta es que hay que fijarse que NFServer implementa Runnable, con lo cual es esta clase la que hay que pasar a un Thread, y run() el metodo que ejecuta el Thread, siendo este declarado efectivamente en NFControllerP2P.

Ejemplo de TODO confuso e incongruente:

- NFServer implements Runnable: "Añadir métodos a esta clase para: [...] 2) Detener el servidor (stopserver)" Como vas a parar el servidor dentro del mismo thread, si el thread está bloqueado en accept()? Si no se usan sockets no bloqueantes (no los hemos visto) la única forma de detener el thread es terminándolo a la fuerza desde el thread Main.

Ejemplo de TODO directamente incorrecto:

- NFServer.serveFilesToClient: "Los métodos lookupHashSubstring y lookupFilenameSubstring de la clase FileInfo lookupHashSubstring no existe en ningún sitio. Además, estas utilidades deberían estar en la base de datos FileDatabase, literalmente son queries a la base de datos. Sin embargo, lookupFilePath sí que está en la FileDatabase, por qué? Es como si se hubieran puesto los metodos de forma aparentemente aleatoria por el código. Nota: esta función tampoco no sirve un proposito porque en el servidor se necesita el resto de campos del objeto FileInfo, no solo su path. Debería de haber un solo metodo para buscar en la db por hash o por nombre y devolver el FileInfo, solo que por alguna razón divinamente desconocida, a FileInfo le falta el getter del path. (Nota: solo tiene sentido guardar el path entero en FileInfo, ya que el nombre es parte del path y por tanto redundante)

FileDatabase es AdHoc, solo sirve como base de datos del cliente:

- escanea un directorio
- no puede guardar peers para usarse en NFDirectoryServer

DirMessage intenta representar todos los tipos de mensajes, haciendo la implementación terrible: encima de que estamos en un lenguaje OOP esto debería ser resuelto por herencia para cada tipo de mensaje

NFDirectoryServer termina inmediatamente al recibir un mensaje mal formado, esto es una vulnerabilidad de Denial of Service

NanoFiles (cliente) termina inmediatamente en cuanto un comando falla. Cuando haces un ping y el directorio no responde, termina, en vez de dejarte volver a intentar

NFController.processCommand()[COM_DOWNLOAD]: A dos metodos aislados se le pasa la misma información inutil (una substring del nombre del archivo). Sin modificar estos, los dos metodos necesitan el hash del archivo -> tendrían que hacer la misma request al directorio para ver a que hash se corresponde (información util) para poder preguntar por A. los servidores y B. a los peers, los chunks del archivo. Diseño fundamentalmente defectuoso. Lo primero que se debería hacer es obtener la información util, el hash, que es con lo que se debe trabajar.