DOCUMENTACIÓN DEL PROYECTO DE PRÁCTICAS - NANOFILES

Daniel Pérez Gómez y Víctor Carrillo Gil
Subgrupo 2.3
Profesor: Eduardo S. Iniesta Soto

ÍNDICE

Introducción

En este documento se especifica el diseño de los protocolos de comunicación entre un cliente y un servidor de ficheros. Además, se definen los diferentes formatos de los mensajes producidos derivados de las conversaciones en el protocolo. Encontraremos cómo interactúan ambas partes bajo un protocolo u otro. También veremos el intercambio de mensajes por cada uno de los lados de la conexión.

Por último, se adjuntan los autómatas desarrollados junto a ejemplos de conversaciones mediante sus usos.

Formato de los mensajes del protocolo de comunicación con el Directorio

Para definir el protocolo de comunicación con el Directorio, vamos a utilizar mensajes textuales con formato **campo:valor** El valor que tome el campo **operation** (código de operación) indicará el tipo de mensaje y por tanto su formato (qué campos vienen a continuación).

Tipos y descripción de los mensajes

CLIENTE → DIRECTORIO

Mensaje: ping
Sentido de la comunicación: Cliente → Directorio
Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para comprobar que el servidor de directorio está activo y que usa un protocolo compatible con el peer.
Ejemplo:

operation: ping
protocol: protocolId

Respuesta:

Directory received datagram from /127.0.0.1:35917 of size 36 bytes. Ping. received protocolId: 123456789A
Sending response operation:pingOk

Mensaje: filelist
Sentido de la comunicación: Cliente → Directorio
Descripción: Solicita los ficheros compartidos en el Directorio.
Ejemplo:

operation: filelist

Respuesta:

These are the files tracked by the directory at localhost
Name Size Hash

Mensaje: serve
Sentido de la comunicación: Cliente → Directorio
Descripción: Solicita que el servidor pueda escuchar conexiones en un puerto efímero
Ejemplo:

operation: serve

Respuesta:

Server started on ephemeral port: 58783
NFServer running on 0.0.0.0:58783
Receiving...operation:serveOk
* File server successfully registered with the directory

Respuesta:

* Downloading file: Elaboración presupuesto proyecto.pdf
* Size: 141033 bytes
* Hash: 449e56e023481ed51f2a5a4c7e27c3220bd563a5
* Downloaded chunk from server 127.0.0.1 (2,9%)
* Downloaded chunk from server 127.0.0.1 (5,8%)
* Downloaded chunk from server 127.0.0.1 (8,7%)
* Downloaded chunk from server 127.0.0.1 (11,6%)
* Downloaded chunk from server 127.0.0.1 (14,5%)
* Downloaded chunk from server 127.0.0.1 (17,4%)
* Downloaded chunk from server 127.0.0.1 (20,3%)
* Downloaded chunk from server 127.0.0.1 (23,2%)
* Downloaded chunk from server 127.0.0.1 (26,1%)
* Downloaded chunk from server 127.0.0.1 (29,0%)
* Downloaded chunk from server 127.0.0.1 (31,9%)
* Downloaded chunk from server 127.0.0.1 (34,9%)
* Downloaded chunk from server 127.0.0.1 (37,8%)
* Downloaded chunk from server 127.0.0.1 (40,7%)
* Downloaded chunk from server 127.0.0.1 (43,6%)
* Downloaded chunk from server 127.0.0.1 (46,5%)
* Downloaded chunk from server 127.0.0.1 (49,4%)
* Downloaded chunk from server 127.0.0.1 (52,3%)
* Downloaded chunk from server 127.0.0.1 (55,2%)
* Downloaded chunk from server 127.0.0.1 (58,1%)
* Downloaded chunk from server 127.0.0.1 (61,0%)
* Downloaded chunk from server 127.0.0.1 (63,9%)
* Downloaded chunk from server 127.0.0.1 (66,8%)
* Downloaded chunk from server 127.0.0.1 (69,7%)
* Downloaded chunk from server 127.0.0.1 (72,6%)
* Downloaded chunk from server 127.0.0.1 (75,5%)
* Downloaded chunk from server 127.0.0.1 (78,4%)
* Downloaded chunk from server 127.0.0.1 (81,3%)
* Downloaded chunk from server 127.0.0.1 (84,2%)
* Downloaded chunk from server 127.0.0.1 (87,1%)
* Downloaded chunk from server 127.0.0.1 (90,0%)
* Downloaded chunk from server 127.0.0.1 (92,9%)
* Downloaded chunk from server 127.0.0.1 (95,8%)
* Downloaded chunk from server 127.0.0.1 (98,7%)
* Downloaded chunk from server 127.0.0.1 (100,0%)
* Download completed successfully

Mensaje: quit
Sentido de la comunicación: Cliente → Directorio | Cliente → Cliente
Descripción: Finaliza la conexión y sale del programa.
Ejemplo:

operation: quit

Respuesta: Bye.

DIRECTORIO → CLIENTE

Mensaje: pingOk

- Sentido de la comunicación: Directorio → Cliente
- Descripción: Es la respuesta devuelta por el servidor de ficheros indicando que el comando “ping” se procesó de forma exitosa
- Responde a: ping

    operation: pingOk

Mensaje: pingBad

- Sentido de la comunicación: Directorio → Cliente
- Descripción: Es la respuesta devuelta por el servidor de ficheros indicando que el comando “ping” no se procesó correctamente
- Responde a: ping

    operation: pingBad

Mensaje: serveOk

- Sentido de la comunicación: Directorio → Cliente
- Descripción: Es la respuesta devuelta por el servidor de ficheros indicando que el comando “serve” se procesó adecuadamente
- Responde a: serve

    operation: serveOk

Mensaje: filelistOk

- Sentido de la comunicación: Directorio → Cliente
- Descripción: Es la respuesta devuelta, donde se muestra una lista de todos los ficheros y sus datos, por el servidor de ficheros indicando que el comando “filelist” se procesó correctamente
- Responde a: filelist

    operation: filelistOk

    filename:
    filesize:
    filehash:
    /// Siguiente fichero ///
    filename:
    filesize:
    filehash:
    …

Mensaje: servers

- Sentido de la comunicación: Cliente → Directorio
- Descripción: Solicita al directorio la lista de servidores que comparten un fichero específico

```
operation: servers
```

Mensaje: serversOk
Sentido de la comunicación: Directorio → Cliente
Descripción: Es la respuesta devuelta por el directorio con la lista de servidores que comparten el fichero solicitado
Responde a: servers

```
operation: serversOk
```

Mensaje: unregister
Sentido de la comunicación: Cliente → Directorio
Descripción: Solicita dar de baja al servidor de ficheros del directorio
Ejemplo:

```
operation: unregister
```

Mensaje: unregisterOk
Sentido de la comunicación: Directorio → Cliente
Descripción: Es la respuesta devuelta por el directorio confirmando la baja exitosa del servidor
Responde a: unregister

```
operation: unregisterOk
```

Mensaje: searchFile
Sentido de la comunicación: Cliente → Directorio
Descripción: Solicita buscar un fichero por nombre o hash en el directorio
Ejemplo:

```
operation: searchFile
```

Mensaje: searchFileOk
Sentido de la comunicación: Directorio → Cliente
Descripción: Es la respuesta devuelta por el directorio con los resultados de la búsqueda
Responde a: searchFile

operation: searchFileOk

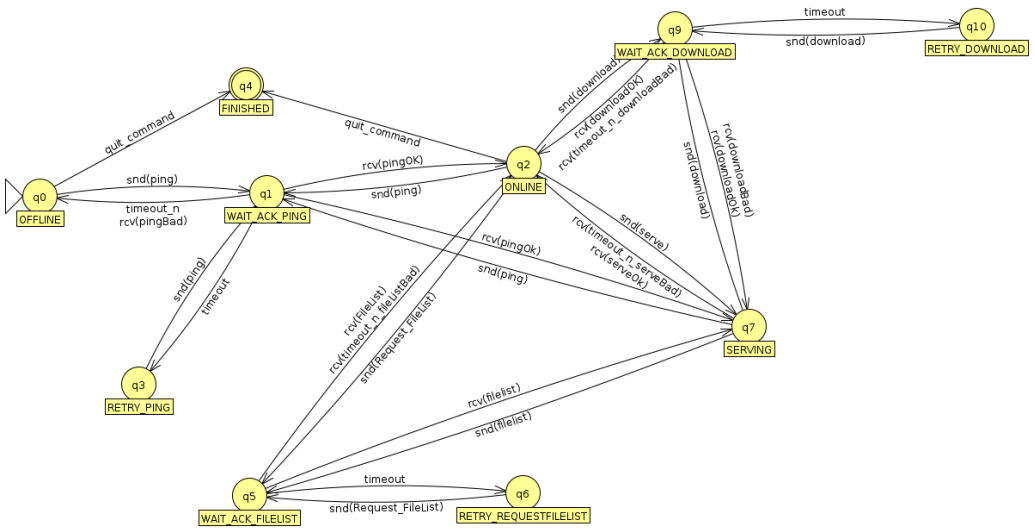
Mensaje: filefound
Sentido de la comunicación: Cliente → Directorio
Descripción: Notifica al directorio que se ha encontrado un fichero específico
Ejemplo:

operation: filefound

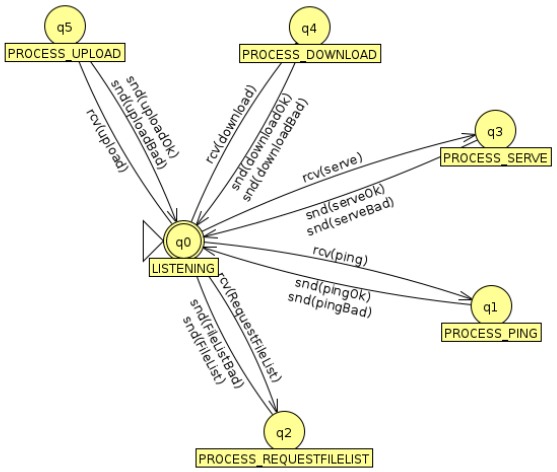
Mensaje: filefoundOk
Sentido de la comunicación: Directorio → Cliente
Descripción: Es la respuesta devuelta por el directorio confirmando la notificación del fichero encontrado
Responde a: filefound
operation: filefoundOk

Autómatas de protocolo

Autómata rol cliente de directorio



Autómata rol servidor de directorio



Formato de los mensajes del protocolo de transferencia de ficheros

Para definir el protocolo de comunicación con un servidor de ficheros, se utilizan mensajes binarios multiformato. El campo opcode indica el tipo de mensaje y su formato.

Tipos y descripción de los mensajes

Mensaje: invalid_opcode (opcode = 0)
Sentido de la comunicación: Servidor → Cliente
Descripción: Este mensaje indica que la operación que se desea realizar no es válida.
Ejemplo:

Opcode (1 byte)
0

Mensaje: downloadFile (opcode = 1)
Sentido de la comunicación: Cliente → Servidor
Descripción: Este mensaje indica la intención de descargar un fichero proporcionando su nombre
Ejemplo:

Opcode (1 byte)	NameLength (4 bytes)	NameBytes (n bytes)

1	**n**	**0x0 0x01 … n**

Mensaje: downloadOk (opcode = 2)
Sentido de la comunicación: Cliente → Servidor
Descripción: Este mensaje indica que se va a iniciar la descarga del fichero mediante el
Ejemplo:

Opcode (1 byte)	NameLength (4 byte)	NameBytes (n bytes)	FileSize (8 bytes)	HashLength (4 byte)	HashBytes (n bytes)
1	**n**	**0x0 0x01 … n**	**n**	**n**	**0x0 0x01 … n**

Mensaje: downloadFail (opcode = 3)
Sentido de la comunicación: Servidor → Cliente
Descripción: Este mensaje indica la operación fallida de descarga, cuando el servidor o bien no encuentra el substring entre sus ficheros o bien cuando el nombre proporcionado es ambiguo.
Ejemplo:

Opcode (1 byte)
3

Mensaje: downloadChunk (opcode = 4)
Sentido de la comunicación: Servidor → Cliente
Descripción: Indica que el cliente solicita al servidor un fragmento (realmente no es necesario porque son de tamaño fijo, pero ya lo había implementado así) concreto de un fichero, indicando la posición inicial (offset) y el tamaño del fragmento. Puede incluir el hash del fichero para mayor seguridad a la hora de la transferencia.

Ejemplo:

Opcode (1 byte)	Offset (4 bytes)	ChunkSize (n bytes)	HashLength (4 bytes)	HashBytes (n bytes)
4	**n**	**0x0 0x01 … n**	**n**	**0x0 0x01 … n**

Mensaje: downloadChunkOk (opcode = 5)
Sentido: Servidor → Cliente
Descripción: Este mensaje indica que se ha descargado el chunk correctamente.
Ejemplo:

Opcode (1 byte)
5

Mensaje: downloadCompleted (opcode = 6)
Sentido: Servidor → Cliente
Descripción: Este mensaje indica que el fichero se ha descargado de forma exitosa.
Ejemplo:

Opcode (1 byte)
6

Mensaje: uploadRequest (opcode = 7)
Sentido: Cliente → Servidor
Descripción: Este mensaje indica que el cliente solicita subir un fichero al servidor, enviando el nombre, el hash y el tamaño del fichero para que el servidor decida si acepta la subida.
Ejemplo:

Opcode (1 byte)	NameSize (2 bytes)	NameData (n bytes)	HashSize (2 bytes)	HashData (n bytes)	FileSize (8 bytes)
7	n	0x0 0x01 … n	n	0x0 0x01 … n	n

Mensaje: uploadOk (opcode = 8)
Sentido: Servidor → Cliente
Descripción: Este mensaje indica que se va a iniciar la subida del fichero al peer correspondiente.
Ejemplo:

Opcode

(1 byte)
8

Mensaje: uploadFail (opcode = 9)
Sentido: Servidor → Cliente
Descripción: Este mensaje indica que no se ha podido realizar correctamente la subida del fichero al peer.
Ejemplo:

Opcode (1 byte)
9

Mensaje: uploadChunk (opcode = 10)
Sentido: Cliente → Servidor
Descripción: Este mensaje indica que el cliente envía un chunk de datos del fichero que está subiendo al servidor, indicando la posición (offset) y el tamaño del fragmento.
Ejemplo:

Opcode (1 byte)	Offset (8 bytes)	Bytes (4 bytes)	Data (n bytes)
1	n	n	0x0 0x01 … n

Mensaje: uploadFinishedOk (opcode = 11)
Sentido: Servidor → Cliente
Descripción: Este mensaje indica que la subida del fichero al peer destino se ha completado existosamente
Ejemplo:

Opcode (1 byte)
11

Mensaje: filelist (opcode = 12)
Sentido: Cliente → Servidor
Descripción: Este mensaje indica que el cliente está solicitando los ficheros compartidos en el directorio.
Ejemplo:

Opcode (1 byte)
12

Mensaje: filelistOk (opcode = 13)
Sentido: Servidor → Cliente
Descripción: Este mensaje indica que la operación filelist se ejecutó correctamente
Ejemplo:

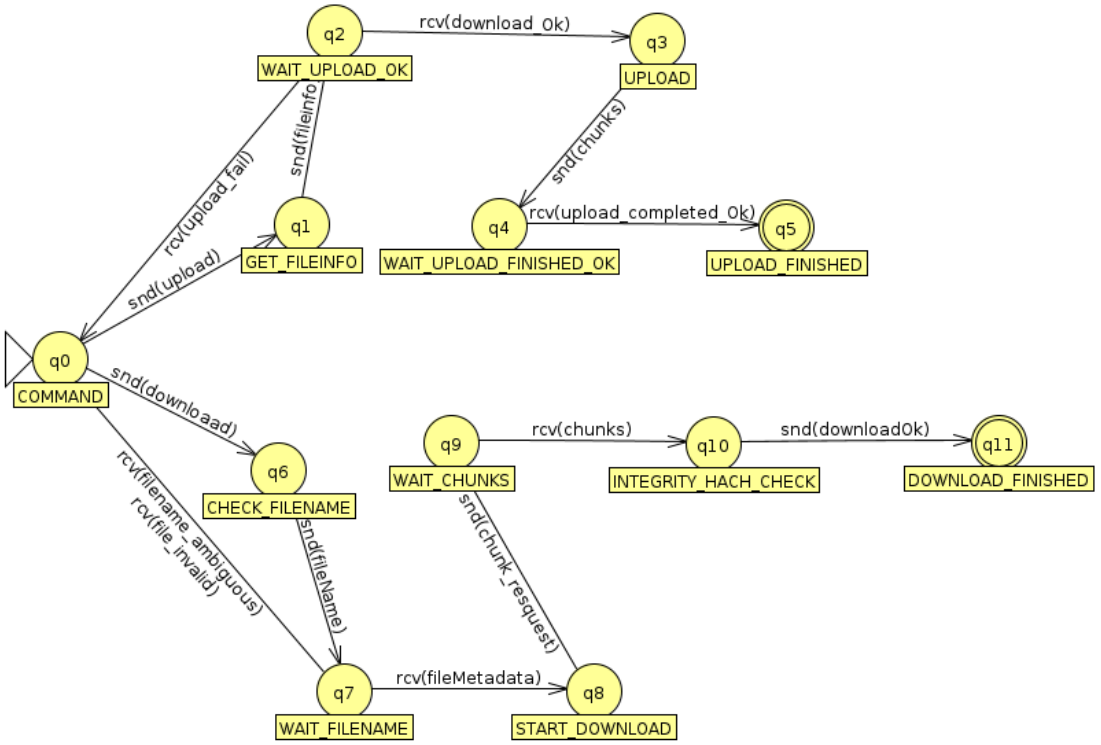
Opcode (1 byte)
13

Mensaje: invalid_operation (opcode = 14)
Sentido: Servidor → Cliente
Descripción: Este mensaje indica que la operación a realizar no es válida
Ejemplo:

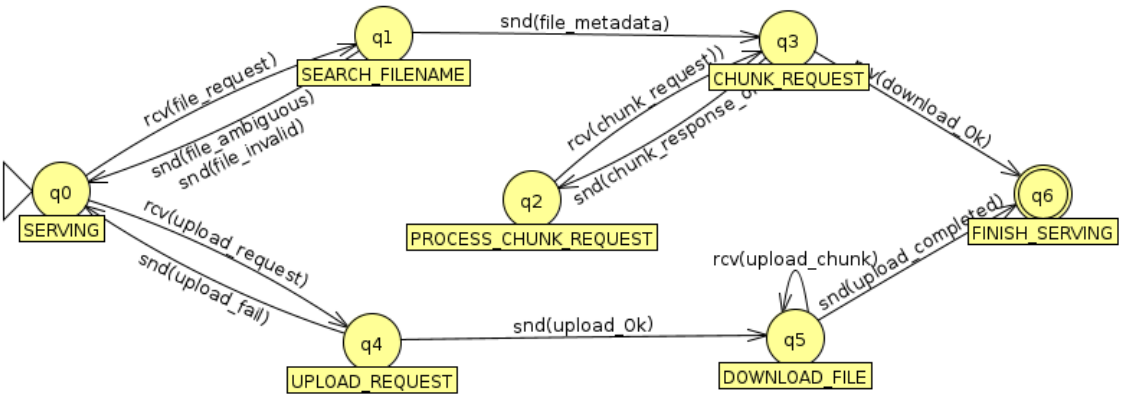
Opcode (1 byte)
14

Autómatas de protocolo

Autómata rol cliente de ficheros



Autómata rol servidor de ficheros



Ejemplo de intercambio de mensajes

Incluye ejemplos de “conversaciones” ficticias usando los mensajes definidos y comentando cómo el autómata restringe qué mensajes puede enviar o recibir cada extremo en cada estado.

Ejemplo: Envío de ping al directorio

CLIENTE (estado: OFFLINE)
operation: ping \n

DIRECTORIO / SERVIDOR DE FICHEROS (estado: LISTENING)
Packet received from /127.0.0.1:57560
Ping received
Receiving...operation:pingOk

CLIENTE (estado: ONLINE)

Ejemplo: Iniciar un servidor de ficheros

CLIENTE (estado: ONLINE)
operation: serve

DIRECTORIO / SERVIDOR DE FICHEROS (estado: ONLINE)
Packet received from /127.0.0.1:57560
Registered 2 files from /127.0.0.1:51564

Server started on ephemeral port: 51564
NFServer running on 0.0.0.0:51564
Receiving...operation:serveOk

CLIENTE (estado: SERVING)

Ejemplo: Solicitud de lista de archivos

CLIENTE (estado: ONLINE)
operation: filelist

DIRECTORIO / SERVIDOR DE FICHEROS(estado: SERVING)
Packet received from /127.0.0.1:57560
Receiving...operation:filelistOk

filename:Ejercicios_Tema_2_solucion.pdf
filesize:77431
filehash:ad5d71a1fdefcf903e688ebeed7f88514d9914a5
filename:Tarea6-Victor.pdf
filesize:95130
filehash:74ed2bb0da5f1fde519bf98fca5f347b020479a2

getFileList: Lista de ficheros obtenida correctamente.
* These are the files tracked by the directory at localhost
Name Size Hash
Ejercicios_Tema_2_solucion.pdf 77431 ad5d71a1fdefcf903e688ebeed7f88514d9914a5
Tarea6-Victor.pdf 95130 74ed2bb0da5f1fde519bf98fca5f347b020479a2

CLIENTE (estado: ONLINE)

Ejemplo: Descarga de un archivo

CLIENTE (estado: ONLINE)
operation: download Ejer ejercicios.pdf

DIRECTORIO / SERVIDOR DE FICHEROS (estado: DOWNLOADING)
operation: send_hash
hash: 5f4dcc3b5aa765d61d8327deb882cf99

CLIENTE (estado: DOWNLOAD_OK)
operation: download_complete

Mejoras implementadas y breve descripción

Comando `upload`

Se ha implementado el comando **upload**, que permite a un peer subir uno de sus ficheros compartidos a otro servidor de ficheros de la red. El funcionamiento es el siguiente:

1. El peer origen selecciona un fichero de su lista de compartidos y solicita subirlo a otro servidor especificando su dirección y puerto.
2. El servidor de destino acepta el fichero **solo si no lo tiene ya** entre sus ficheros disponibles (la comprobación se realiza por nombre, tamaño y hash).
3. La transferencia se realiza en bloques binarios (chunks) usando TCP, aprovechando la fiabilidad y el orden de entrega del protocolo.
4. Al finalizar la transferencia, el servidor verifica la integridad del fichero recibido mediante el hash y confirma la subida exitosa al peer origen.

La lógica se ha implementado añadiendo nuevos opcodes y mensajes binarios en el protocolo P2P (OPCODE_UPLOAD_REQUEST, OPCODE_UPLOAD_OK, OPCODE_UPLOAD_CHUNK, OPCODE_UPLOAD_FINISHED_OK, etc.), así como los métodos necesarios en las clases PeerMessage, PeerMessageOps, NFConnector, NFControllerLogicP2P y NFServer.
El servidor solo acepta la subida si el fichero no existe ya en su base de datos, y la transferencia se realiza de forma eficiente y segura, garantizando la integridad del fichero recibido.

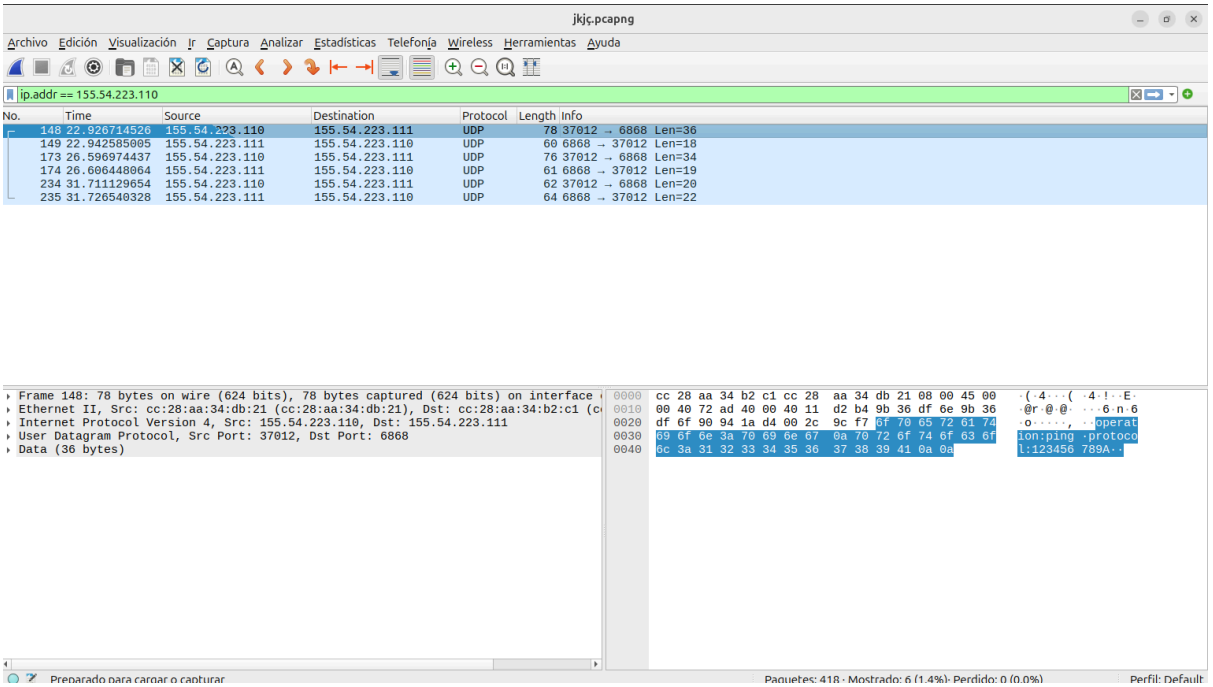
Comando `serve` con puerto efímero

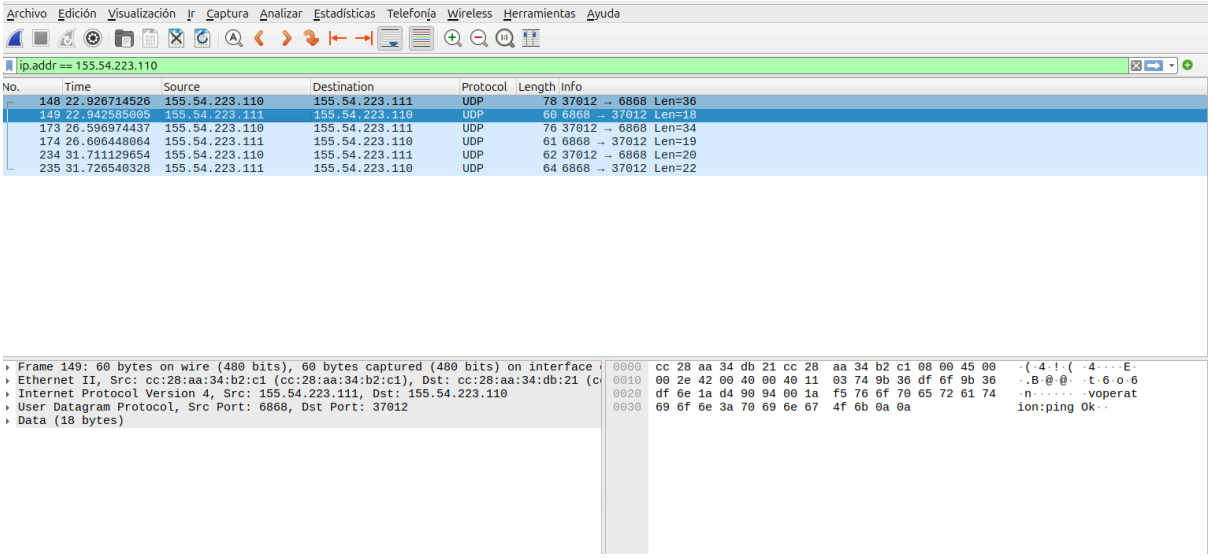
Se ha ampliado el comando **serve** para que el servidor de ficheros pueda utilizar **cualquier puerto disponible** (puerto efímero) para escuchar conexiones entrantes, en lugar de estar limitado al puerto fijo 10000/tcp

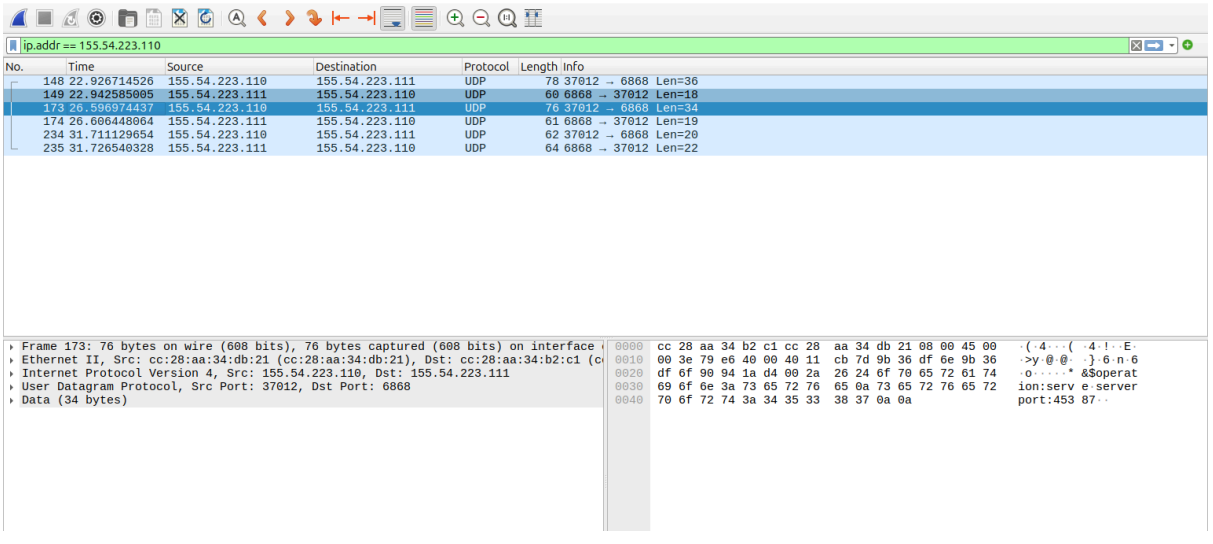
El servidor de ficheros (NFServer) se inicializa ahora con el puerto 0, lo que indica al sistema operativo que asigne automáticamente un puerto libre (puerto efímero). El puerto asignado se recupera mediante getLocalPort() y se comunica al directorio y a los peers para que puedan conectarse correctamente.
Esta mejora permite ejecutar múltiples istancias del servidor en la misma máquina y así evitar conflictos por el uso de puertos, haciendo el sistema más robusto.

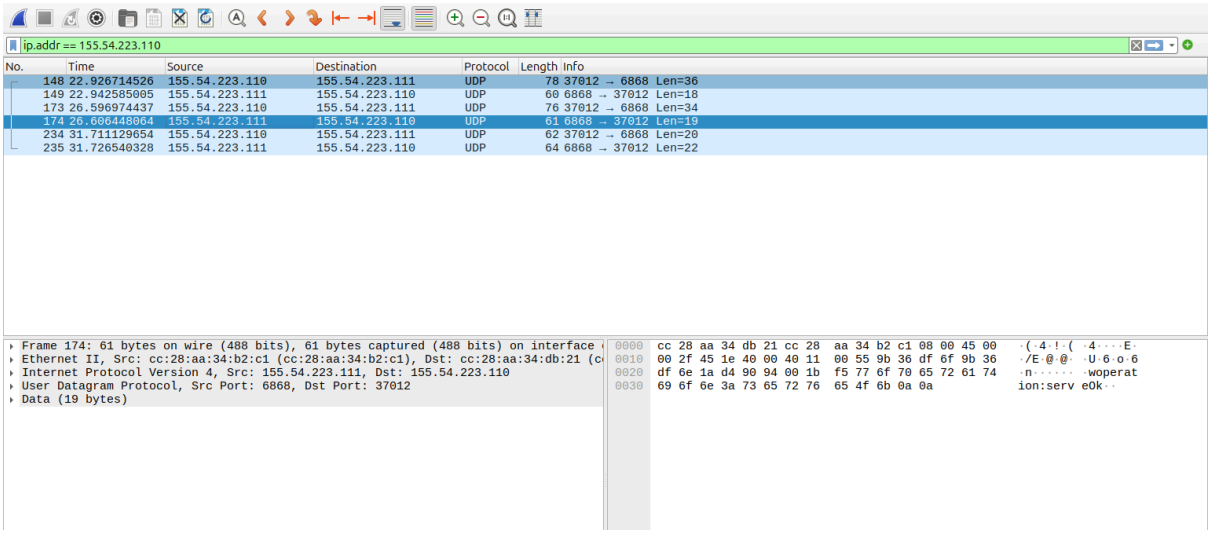
Capturas en Wireshark

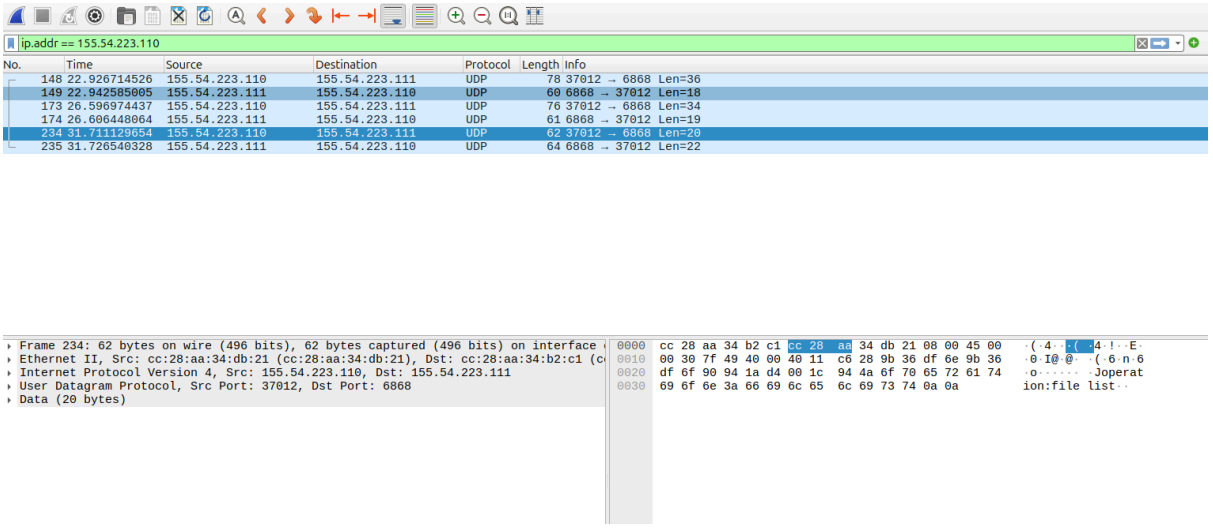
A continuación muestro 5 imágenes de mensajes UDP capturados mediante Wireshark.


jkjç.pcapng
Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda
ip.addr == 155.54.223.110
No. Time Source Destination Protocol Length Info
148 22.926714526 155.54.223.110 155.54.223.111 UDP 78 37012 → 6868 Len=36
149 22.942585005 155.54.223.111 155.54.223.110 UDP 60 6868 → 37012 Len=18
173 26.596974437 155.54.223.110 155.54.223.111 UDP 76 37012 → 6868 Len=34
174 26.606448064 155.54.223.111 155.54.223.110 UDP 61 6868 → 37012 Len=19
234 31.711129654 155.54.223.110 155.54.223.111 UDP 62 37012 → 6868 Len=20
235 31.726540328 155.54.223.111 155.54.223.110 UDP 64 6868 → 37012 Len=22
Frame 148: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface
Ethernet II, Src: cc:28:aa:34:db:21 (cc:28:aa:34:db:21), Dst: cc:28:aa:34:b2:c1 (c
Internet Protocol Version 4, Src: 155.54.223.110, Dst: 155.54.223.111
User Datagram Protocol, Src Port: 37012, Dst Port: 6868
Data (36 bytes)
0000 cc 28 aa 34 b2 c1 cc 28 aa 34 db 21 08 00 45 00
0010 00 40 72 ad 40 00 40 11 d2 b4 9b 36 df 6e 9b 36
0020 df 6f 90 94 1a d4 00 2c 9c f7 6f 70 65 72 61 74
0030 69 6f 6e 3a 70 69 6e 67 0a 70 72 6f 74 6f 63 6f
0040 6c 3a 31 32 33 34 35 36 37 38 39 41 0a 0a
Preparado para cargar o capturar
Paquetes: 418 · Mostrado: 6 (1.4%) · Perdido: 0 (0.0%)
Perfil: Default


Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda
ip.addr == 155.54.223.110
No. Time Source Destination Protocol Length Info
148 22.926714526 155.54.223.110 155.54.223.111 UDP 78 37012 → 6868 Len=36
149 22.942585005 155.54.223.111 155.54.223.110 UDP 60 6868 → 37012 Len=18
173 26.596974437 155.54.223.110 155.54.223.111 UDP 76 37012 → 6868 Len=34
174 26.606448064 155.54.223.111 155.54.223.110 UDP 61 6868 → 37012 Len=19
234 31.711129654 155.54.223.110 155.54.223.111 UDP 62 37012 → 6868 Len=20
235 31.726540328 155.54.223.111 155.54.223.110 UDP 64 6868 → 37012 Len=22
Frame 149: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface
Ethernet II, Src: cc:28:aa:34:b2:c1 (cc:28:aa:34:b2:c1), Dst: cc:28:aa:34:db:21 (c
Internet Protocol Version 4, Src: 155.54.223.111, Dst: 155.54.223.110
User Datagram Protocol, Src Port: 6868, Dst Port: 37012
Data (18 bytes)
0000 cc 28 aa 34 db 21 cc 28 aa 34 b2 c1 08 00 45 00
0010 00 2e 42 00 40 00 40 11 03 74 9b 36 df 6f 9b 36
0020 df 6e 1a d4 90 94 00 1a f5 76 6f 70 65 72 61 74
0030 69 6f 6e 3a 70 69 6e 67 4f 6b 0a 0a


ip.addr == 155.54.223.110
No. Time Source Destination Protocol Length Info
148 22.926714526 155.54.223.110 155.54.223.111 UDP 78 37012 → 6868 Len=36
149 22.942585005 155.54.223.111 155.54.223.110 UDP 60 6868 → 37012 Len=18
173 26.596974437 155.54.223.110 155.54.223.111 UDP 76 37012 → 6868 Len=34
174 26.606448064 155.54.223.111 155.54.223.110 UDP 61 6868 → 37012 Len=19
234 31.711129654 155.54.223.110 155.54.223.111 UDP 62 37012 → 6868 Len=20
235 31.726540328 155.54.223.111 155.54.223.110 UDP 64 6868 → 37012 Len=22
Frame 173: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface
Ethernet II, Src: cc:28:aa:34:db:21 (cc:28:aa:34:db:21), Dst: cc:28:aa:34:b2:c1 (c
Internet Protocol Version 4, Src: 155.54.223.110, Dst: 155.54.223.111
User Datagram Protocol, Src Port: 37012, Dst Port: 6868
Data (34 bytes)
0000 cc 28 aa 34 b2 c1 cc 28 aa 34 db 21 08 00 45 00
0010 00 3e 79 e6 40 00 40 11 cb 7d 9b 36 df 6e 9b 36
0020 df 6f 90 94 1a d4 00 2a 26 24 6f 70 65 72 61 74
0030 69 6f 6e 3a 73 65 72 76 65 0a 73 65 72 76 65 72
0040 70 6f 72 74 3a 34 35 33 38 37 0a 0a





Enlace a grabación de vídeo de pruebas

https://youtu.be/ebJsmaK6Mcg

Conclusiones y valoraciones personales

En primer lugar, empezaremos comentando las dificultades del proyecto y terminaremos con nuestras opiniones personales.

En cuanto al proyecto, creemos que es interesante la idea de “crear” una aplicación mediante la que podamos hacer uso de operaciones tan frecuentes en la vida cotidiana como lo son la descarga y la subida de ficheros.

Cuando acabamos el proyecto sentimos una gran satisfacción al poder haber implementado en gran medida los requisitos exigidos e incluso alguna mejora, pero no todo fue sencillo. Desde nuestro punto de vista, el proyecto tiene varios inconvenientes, empezando por el lenguaje, pues es un lenguaje muy nuevo para nosotros, ya que el contacto con él comenzó hace apenas unos meses en la asignatura de POO, por lo que ya existe una dependencia de conocimientos previos, especialmente por Java, al ser un lenguaje orientado a objetos, algo totalmente novedoso para alumnos de segundo curso y que puede ser complicado si el alumno no entiende bien el funcionamiento de las clases, métodos y la concepción de los objetos en sí. A pesar de esto, este no ha sido el principal problema, pues también es resulta costoso adaptarse al código dado inicialmente, ya que ni las variables, ni métodos, ni objetos han sido implementados por nosotros y requiere una adaptación a la hora de entender qué función tiene cada variable y de qué tipo es, lo que conlleva una demora en el tiempo de trabajo de cara a la implementación.

Dicho esto, el principal inconveniente de este proyecto es sin duda su gran extensión y densidad, pues es un proyecto que requiere invertir una gran cantidad de horas para su realización, que se suma también a otros proyectos coincidentes en el mismo período de tiempo como lo son los proyectos de las asignaturas de Compiladores y AED II, lo que dificulta seguir un ritmo constante a pesar de asistir a clases de prácticas.

En lo que respecta a nuestra opinión final, podemos afirmar que este proyecto es hasta ahora el más difícil con diferencia contra el que nos hemos enfrentado. Igual que antes hemos comentado los problemas e inconvenientes de este trabajo, también nos ha aportado beneficios y conocimientos en nuestro desarrollo como ingenieros, pues es esencial trabajar desde pronto y entender cómo funciona la red, y qué mejor forma de hacerlo que mediante el desarrollo de una aplicación Cliente-Servidor o P2P. Nos parece una forma interesante y dinámica de aprender estos contenidos, obviando su dificultad y extensión. También hemos aprendido a manejar de forma avanzada el entorno de Eclipse IDE, pues durante el desarrollo hemos experimentado con errores y las horas invertidas en depurar código y aprender técnicas como el uso de breakpoints son muy útiles de cara a futuros proyectos. Esto ha sido un proyecto duro, pero ver que obtenemos un resultado real y visible nos hace valorar el esfuerzo realizado y ver que proyectos como este (modificando y adaptándose correctamente al tiempo del que disponemos) son realmente útiles para las siguientes promociones de alumnos de Ingeniería Informática.