REDES DE COMUNICACIONES

Proyecto NanoFiles

Participantes:

Abdellah Bensaidi Souri (Subgrupo 1.2)
Roberto Ayllón Zamora (Subgrupo 1.2)

Índice:

1.- Introducción **3**
2.- Formato de los mensajes del protocolo de comunicación con el Directorio **4**
3.-Formato de los mensajes del protocolo de transferencia de ficheros **6**
3.1.-Formatos de mensajes 6
3.2.-Tipos y descripción de los mensajes binarios 7
4.- Autómatas **8**
4.1.-Autómata rol cliente de directorio 8
4.2.-Autómata rol servidor de directorio 9
4.3.-Autómata rol cliente de ficheros 9
4.4.-Autómata rol servidor de ficheros 10
5.-Mejoras implementadas **10**
6.-Ejemplo de intercambio de mensajes **13**
7.-Conclusiones **15**

1.- Introducción

Este informe presenta el diseño del proyecto NanoFiles, un sistema que emplea los protocolos UDP y TCP para establecer comunicación entre clientes y servidores con el objetivo de compartir ficheros.

En cuanto a la comunicación UDP, dado que este protocolo no garantiza la entrega de mensajes, hemos desarrollado un conjunto de mensajes con sus respectivas confirmaciones (como el par "ping" y "ping_success" que confirma la recepción correcta). Para situaciones más complejas, como las peticiones de descarga mediante "download_request", hemos incluido respuestas de error específicas: enviamos "download_fail" cuando la subcadena del fichero solicitado se corresponde con múltiples ficheros (resultado ambiguo) o cuando no existe ninguna coincidencia en los servidores disponibles. Este intercambio de mensajes ocurre entre el directorio del cliente y el directorio del servidor.

Respecto a la comunicación TCP, garantizamos una transmisión fiable mediante la implementación de mensajes binarios que facilitan el intercambio entre peers (donde uno actúa como cliente solicitante y otro como servidor de ficheros). Hemos diseñado formatos específicos para cada tipo de mensaje según la operación requerida, logrando así una comunicación más eficiente.

Además, se han adjuntado en el documento diversos autómatas que muestran cómo es la comunicación cliente-servidor desde cada perspectiva, facilitando la comprensión del funcionamiento del protocolo puesto que es más visual.

En la sección de mejoras detallamos todas las funcionalidades añadidas: tanto las obligatorias como funcionalidad mínima para la convocatoria de junio/julio como otras adicionales, como una barra de progreso para las descargas, la implementación del comando 'stopserver' para detener el servidor, y el almacenamiento automático en la base de datos local de los ficheros descargados, entre otras.

Finalmente, incluimos un apartado con capturas de Wireshark que enseña el intercambio real de mensajes entre clientes y servidores.

2.- 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

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

```
operation: ping\n
protocol: 123456789A\n
\n
```

Mensaje: **ping_success**
Sentido de la comunicación: Directorio→ Cliente
Descripción: Este mensaje lo envía el Directorio de NanoFiles al cliente para que le confirme que lo escucha y que utiliza un protocolo compatible con él.
Ejemplo:

```
operation: ping_success\n
protocol: 123456789A\n
\n
```

Mensaje: **ping_fail**
Sentido de la comunicación: Directorio→ Cliente
Descripción: Este mensaje lo envía el Directorio de NanoFiles al peer para informarle que no lo escucha y que no utiliza un protocolo compatible con este.
Ejemplo:

```
operation: ping_fail\n
\n
```

Mensaje: **serve**
Sentido de la comunicación: Cliente → Directorio
Descripción: Lanza un servidor de ficheros que escucha conexiones en cualquier puerto. Al lanzar un servidor de ficheros, se publica automáticamente al directorio los metadatos (nombre, tamaño y hash) de los ficheros que este peer tiene en su carpeta compartida.
Ejemplo:

```
operation: serve\n
port: 12345\n
names: nombre1, nombre2, …, nombren\n
sizes: tamaño1, tamaño2, …, tamañon,\n
hashes: hash1, hash2, …, hashn\n
\n
```

Mensaje: **serve_success**
Sentido de la comunicación: Directorio → Cliente
Descripción: Este mensaje lo envía el Directorio al peer para confirmar que se ha registrado correctamente como servidor de archivos y sus ficheros han sido añadidos al directorio.
Ejemplo:

```
operation: serve_success\n
\n
```

Mensaje: **get_filelist**
Sentido de la comunicación: Cliente → Directorio
Descripción: El cliente solicita al directorio que se muestren los ficheros que han sido compartidos por otros peers, que han sido publicados en el directorio.
Ejemplo:

```
operation: get_filelist\n
\n
```

Mensaje: **filelist**
Sentido de la comunicación: Directorio → Cliente
Descripción: Muestra los ficheros que han sido compartidos por otros peers y publicados en el directorio como respuesta al mensaje “get_filelist”.
Ejemplo:

```
operation: filelist\n
names: nombre_1, nombre_2, …, nombre_n\n
sizes: tamaño_1, tamaño_2, …, tamaño_n,\n
hashes: hash_1, hash_2, …, hash_n\n
servers: {servidores_1}, {servidores_2}, …, {servidores_n}\n
\n
```

Donde $servidores_i$ representa una lista de servidores asociada al fichero número i cuyo formato **/IP:PORT**.

Mensaje: **download_request**
Sentido de la comunicación: Directorio → Cliente
Descripción: En caso de que el mensaje anterior haya resultado exitoso, proceder con la descarga.
Ejemplo:

```
operation: download_request\n
filename_substring: subcadena\n
\n
```

Mensaje: **download**
Sentido de la comunicación: Cliente → Directorio
Descripción: Tenemos un string llamado “filename_substring”. Obtiene todos los servidores que tengan el archivo cuyo nombre coincida con “filename_substring” y a partir de ahí descarga el archivo.
Ejemplo:

```
operation: download\n
filename_substring: subcadena\n
servers: servidor_1, servidor_2, servidor_3, …, servidor_n\n
\n
```

Mensaje: **download_fail**
Sentido de la comunicación: Directorio → Cliente
Descripción: Informa de que la descarga no se ha podido realizar.
Ejemplo:

```
operation: download_fail\n
\n
```

Mensaje: **unpublish_files**
Sentido de la comunicación: Cliente → Directorio
Descripción: Mensaje que provoca la salida del programa, y en caso de haber ejecutado previamente el comando “serve”, eliminará todos sus archivos del directorio junto con su servidor, por eso se pasa el servidor que se quiere dar de baja.
Ejemplo:

```
operation:unpublish_files\n
port: 12345\n
\n
```

Mensaje: **unpublish_files_success**
Sentido de la comunicación: Directorio → Cliente
Descripción: Confirmación de que se ha ejecutado con éxito.
Ejemplo:

```
operation: unpublish_files_success \n
\n
```

3.-Formato de los mensajes del protocolo de transferencia de ficheros

Para definir el protocolo de comunicación con un servidor de ficheros, vamos a utilizar mensajes binarios multiformato. El valor que tome el campo **opcode** (código de operación) indicará el tipo de mensaje y por tanto cuál es su formato, es decir, qué campos vienen a continuación.

3.1.-Formatos de mensajes

Formato: Control

Opcode
1 byte

Formato: Operación

Opcode	Parametro1	Parametro2
1 byte	8 bytes	8 byte

Formato: TLV (Tipo-Longitud-Valor)

Opcode	Longitud	Valor
1 byte	1/2/4/8 bytes	n bytes

3.2.-Tipos y descripción de los mensajes binarios

Mensaje: fileDownloadChunk (opcode = 1)
Formato: TLV
Sentido de la comunicación: Cliente → Servidor de ficheros
Descripción: Este mensaje es enviado para solicitar la descarga de un fichero. En dicho mensaje se adjuntan los datos del fichero que se quieren descargar, como su hash, el offset (por así decirlo, el número del último byte descargado del fichero) y el máximo tamaño del trozo que queremos descargar del fichero, entre otros. Ejemplo:

Opcode (1 byte)	Longitud (1 byte)	Valor (n bytes)
1	x	xxxxxx…

Mensaje: fileNotFound (opcode = 2)
Formato : Control
Sentido de la comunicación: Servidor de ficheros → Cliente
Descripción: Este mensaje lo envía el peer servidor de ficheros al peer cliente (receptor) de fichero para indicar que no es posible encontrar el fichero con la información proporcionada en el mensaje de petición de descarga.
Ejemplo:

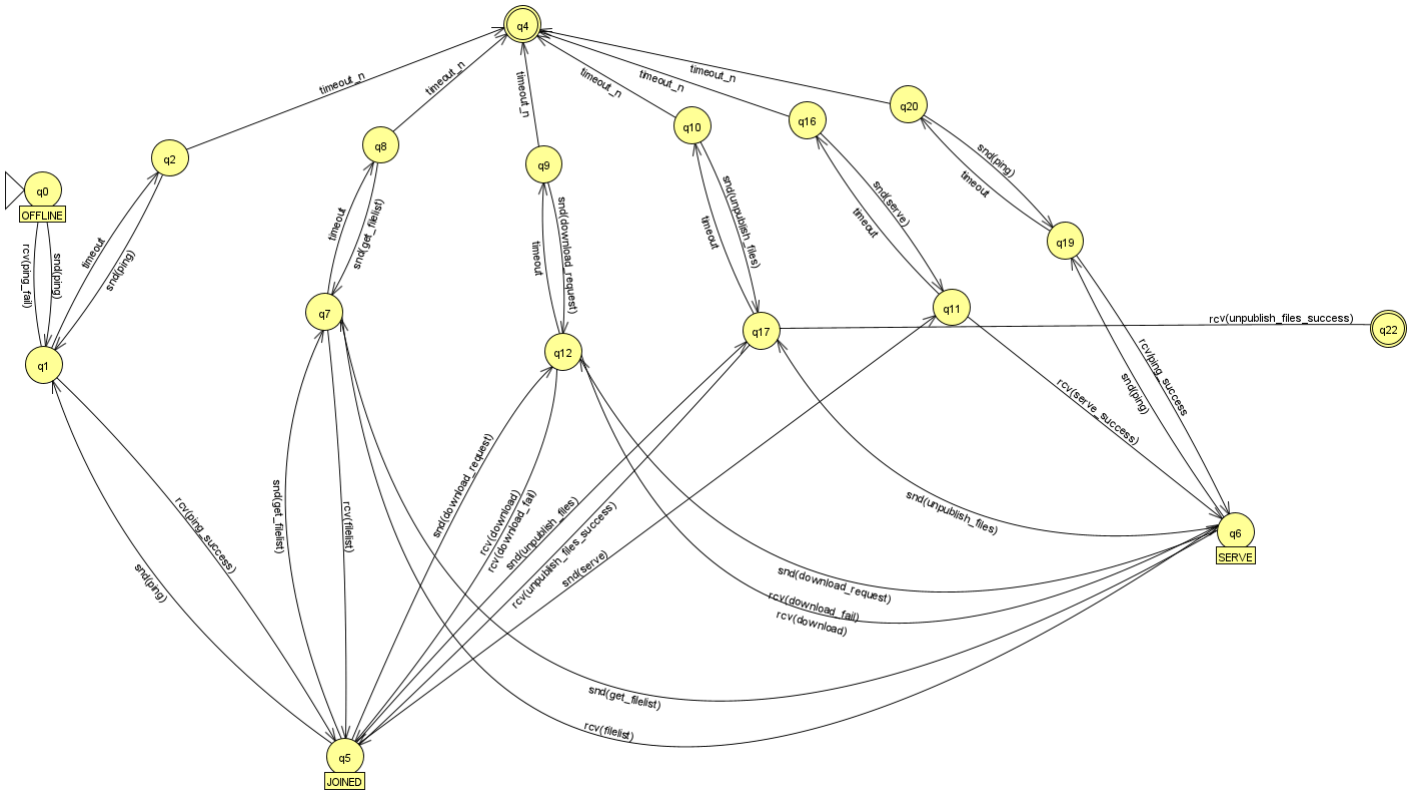
Opcode (1 byte)
2

Mensaje: fileChunk (opcode = 3)
Formato: TLV
Sentido de la comunicación: Servidor de ficheros → Cliente
Descripción: Este mensaje es enviado en respuesta a la solicitud de descarga de un chunk de un fichero. El valor asignado al `OPCODE` es 3. También mandaremos datos del fichero buscado, en especial, el hash, el tamaño del fichero, y su contenido. Ejemplo:

Opcode (1 byte)	Longitud (1 byte)	Valor (n bytes)
3	x	xxxxxx...

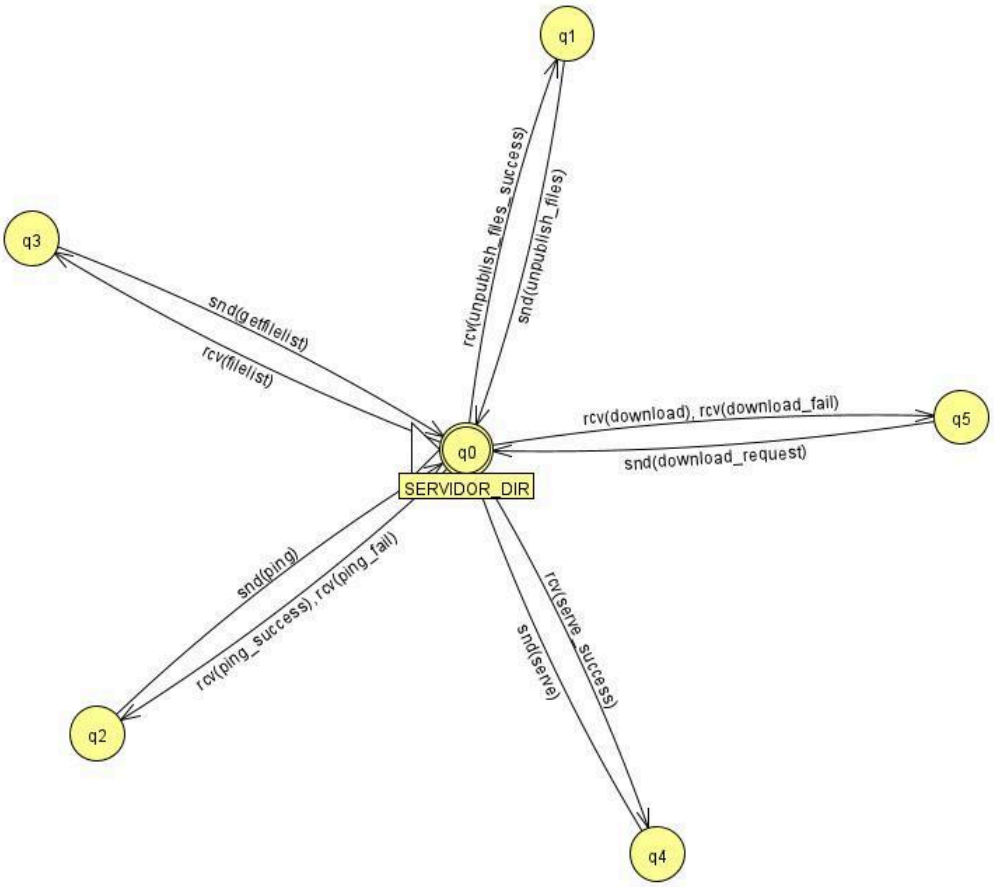
4.- Autómatas

4.1.-Autómata rol cliente de directorio

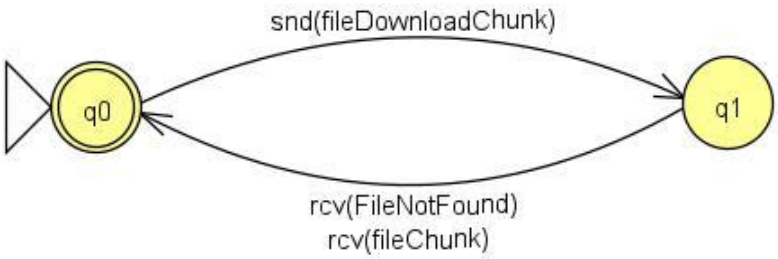


Una restricción que hay que tener en cuenta es que desde `SERVE` se puede ir hasta `JOINED` mediante el mensaje de confirmación `unpublish_files_success`, pero también desde dicho mensaje se puede ir al estado final. Esto se debe a que al ejecutar el comando `quit` y al ejecutar el comando `stopserver` se envía el mismo mensaje `unpublish_files`, por lo que dependiendo del comando se irá a un estado u a otro. Desde `SERVE` se irá al estado final con el mensaje `unpublish_files` que ejecuta el comando `QUIT`, y se irá al estado `JOINED` al enviar el mensaje `unpublish_files` desde el comando `STOPSERVER`. Desde `JOINED` solo se podrá ir al estado final una vez ejecutado el comando `QUIT`, no puede acceder a ningún otro estado en caso de enviar el mensaje `unpublish_files` desde el comando `STOP_SERVER`.

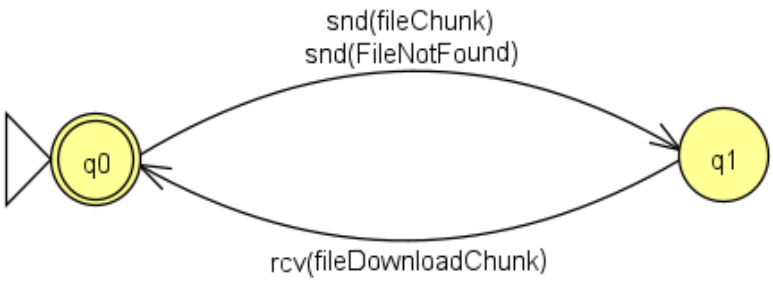
4.2.-Autómata rol servidor de directorio



4.3.-Autómata rol cliente de ficheros



4.4.-Autómata rol servidor de ficheros



5.-Mejoras implementadas

Las tres primeras mejoras que se van a explicar son las que se implementan como la funcionalidad mínima obligatoria, con las que se podrá optar a un máximo de 7 puntos. El resto de mejoras son ampliaciones para subir nota.

`Serve con puerto efímero:` Para permitir la creación de servidores con puertos que se asignen automáticamente, se ha modificado la clase `NFServer` añadiendo un nuevo constructor que acepta un parámetro de puerto. Cuando se desea utilizar un puerto efímero, simplemente hay que pasar el valor 0 al constructor. Esta funcionalidad está implementada en la clase `NFControllerLogicP2P`, donde se utiliza el puerto 0 para que el sistema operativo elija dinámicamente un puerto libre.

`Comando quit actualiza ficheros y servidores:` Hemos ampliado el método `unregisterFileServer` de la clase `DirectoryConnector` para que funcione de manera similar a otros métodos del sistema como `pingDirectory()` o `registerFileServer()`. Básicamente, envía un mensaje `unpublish_files` y espera recibir la confirmación `unpublish_files_success`, devolviendo **`true`** en caso de haberlo recibido y **`false`** en caso contrario.

Cuando el mensaje llega a la clase `NFDirectoryServer`, esta se encarga de limpiar todas las referencias a los ficheros del servidor que se está dando de baja, eliminándolos de las estructuras de datos `ficherosDisponibles` y `ficherosEnServidor`. Así, la próxima vez que alguien ejecute el comando `filelist`, no verá ficheros de un servidor que ya no existe.

A veces puede ocurrir que el mismo fichero (con el mismo hash) esté disponible en varios servidores diferentes. En estos casos, cuando damos de baja uno de los servidores, simplemente actualizamos la información para que el fichero aparezca solo con el nombre que tiene en el servidor que sigue activo. De esta forma no perdemos el fichero, solo actualizamos de dónde viene.

`Comando filelist ampliado con servidores:` Hemos ampliado la funcionalidad del comando `filelist` para que, además de mostrar los metadatos de cada archivo, también indique en qué servidores está disponible. Para implementar esto, se ha modificado el método `sendResponse`. Mientras

recorremos la estructura de datos `ficherosDisponibles`, para cada archivo tomamos su hash único y consultamos la estructura `ficherosEnServidor` para ver en qué servidores está alojado ese mismo archivo. De esta manera, podemos construir una lista completa de servidores para cada fichero, que se expresará de la forma $\{servidor_1, servidor_2, \ldots, servidor_n\}$ para el i-ésimo fichero.

Comando stopserver, que detiene el servidor de ficheros: En la clase `NFCommands`, se ha implementado este nuevo comando basándonos en la lógica de los demás comandos, añadiéndolo a las estructuras de datos y métodos que se encuentran en dicha clase. Con respecto a la clase `NFController`, se ha utilizado una lógica similar a la del comando quit, con la excepción de que al ejecutar este comando el programa sigue en ejecución.

Actualizar automáticamente la base de datos local cada vez que se descargue un fichero: La implementación de esta mejora es la más sencilla de todas, bastó con añadir en el método este fragmento de código:

```
long fileSize = outputFile.length();
FileInfo fileInfo = new FileInfo(hashFileCalculated, localFileName, fileSize,
outputFile.getPath());
NanoFiles.db.addFile(fileInfo);
```

que se ejecutará si el hash del fichero obtenido mediante mensajes es equivalente al hash obtenido con el método estático `computeFileChecksumString(String …)`.

Ejecutar tanto NanoFiles como Directory con un argumento para obtener mayor verbosidad: Se ha modificado esencialmente las clases `NanoFiles` y `Directory` para que admitan este nuevo argumento, siendo **`--debug-level=x`** donde `x` es un valor entre 0 y 2 (ambos inclusive).

El valor 0 significa que no se va a imprimir ningún mensaje de depuración por pantalla, el valor 1 (que también será el valor por defecto) imprimirá los mensajes esenciales de depuración, y finalmente, el valor 2 imprimirá todos los mensajes de depuración que se han ido poniendo en el proyecto.

Aparte de modificar estas dos clases, se han modificado casi todas las demás en las que hubiese un mensaje de depuración, para añadir la condición de que se muestre o no dicho mensaje en base al argumento recibido.

Mostrar en el prompt de NF el estado actual del autómata: Solo se ha modificado la clase `NFController`, en la que se han añadido tres constantes de tipo byte, que representarán los estados del autómata:

- **`*OFFLINE`**: Representa el estado inicial, en el que aún no se ha ejecutado por primera vez el comando `ping`.
- **`*JOINED`**: Estado al que se llega tras haber ejecutado por primera vez el comando `ping`, o al que se llega a través de cualquier comando que no sea `serve`, ya que este nos llevaría al estado **`SERVE`**.
- **`*SERVE`**: Representa el estado al que se llega a través de la ejecución del comando `serve`. Desde este estado se puede ejecutar cualquier comando que no sea el comando `serve` y solamente se puede salir de este estado tras ejecutar o bien el comando quit o el comando `stopserver`, volviendo al estado **`JOINED`**.

Se ha añadido una variable de tipo `byte` llamada `currentState` representando en qué estado del autómata nos encontramos en cierto instante.

Aparte de esto, se hace una ligera modificación en la clase `NFShell`, en el método `readGeneralCommandFromStdIn()`, añadiendo un switch que mirará la variable `currentState` y en base al estado del autómata escribirá `([OFFLINE] nanoFiles@nf-shared1/)`, `([JOINED] nanoFiles@nf-shared1/)` o `([SERVE] nanoFiles@nf-shared1/)`.

Barra de progreso en descargas: Para añadir la barra de progreso en el proceso de descargar un fichero, se han realizado modificaciones específicas en dos clases: `NFControllerLogicP2P` y `PeerMessage`.

En `NFControllerLogicP2P` se ha creado un método llamado `barraProgreso` que toma tres parámetros esenciales: bytes descargados, tamaño total del archivo y un booleano que será false si la descarga no se ha completado y true en caso contrario. Este método genera una representación visual del progreso y devuelve la cadena formateada correspondiente.

La barra se actualiza en cada iteración del bucle while del método `downloadFileFromServers`. Durante la descarga activa muestra el formato:
`* Downloading: [======== ] offset/file_size bytes -> x%`, donde cada "=" simboliza los bytes que han sido descargados y los espacios representan los bytes pendientes de descargar. El porcentaje se calcula con precisión decimal redondeada a la centésima.

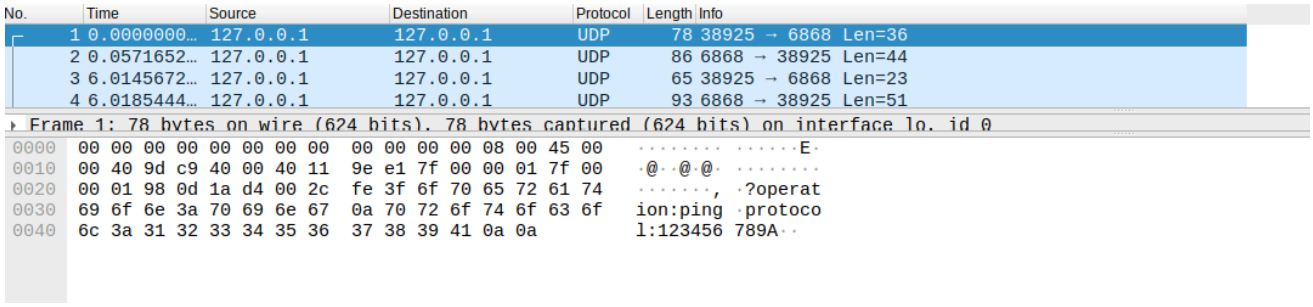
Hacemos uso de un retorno de carro que borra la línea anterior antes de mostrar la actualización. Esta técnica crea la ilusión de una barra que se actualiza suavemente en lugar de generar múltiples líneas de salida.

Al completarse la descarga, el booleano se activa para garantizar la visualización del 100% completo:
`* Downloading: [====================] file_size/file_size bytes -> 100,00%`. La idea de usar un booleano para representar que se ha terminado la descarga para imprimir esta cadena viene de pruebas que hemos realizado con anteriores descargas, en las que hemos comprobado que aunque la variable offset se aproxime bastante al tamaño del fichero, no llega a ser exactamente igual, por lo que lo hemos hecho así para forzar a que se muestre que sí se ha terminado la descarga.

La clase `PeerMessage` se ha actualizado para transmitir el tamaño total del archivo a través de los mensajes `TCP`, proporcionando así el dato fundamental que necesita el método `barraProgreso` para calcular correctamente el porcentaje de avance.

6.-Ejemplo de intercambio de mensajes

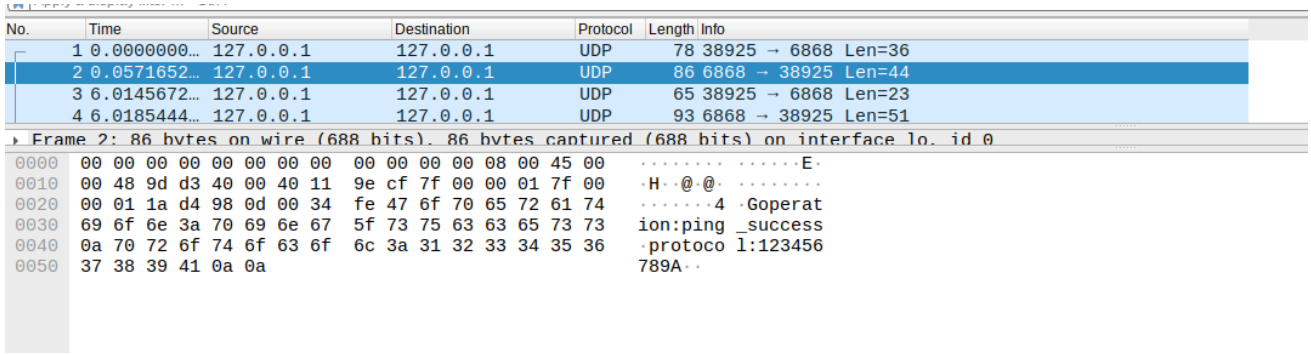
Ejemplo de `ping` y `ping_success`:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000…	127.0.0.1	127.0.0.1	UDP	78	38925 → 6868 Len=36
2	0.0571652…	127.0.0.1	127.0.0.1	UDP	86	6868 → 38925 Len=44
3	6.0145672…	127.0.0.1	127.0.0.1	UDP	65	38925 → 6868 Len=23
4	6.0185444…	127.0.0.1	127.0.0.1	UDP	93	6868 → 38925 Len=51

Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface lo, id 0

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  00 40 9d c9 40 00 40 11  9e e1 7f 00 00 01 7f 00   ·@··@·@· ........
0020  00 01 98 0d 1a d4 00 2c  fe 3f 6f 70 65 72 61 74   .......,  ·?operat
0030  69 6f 6e 3a 70 69 6e 67  0a 70 72 6f 74 6f 63 6f   ion:ping  ·protoco
0040  6c 3a 31 32 33 34 35 36  37 38 39 41 0a 0a         l:123456 789A··
```

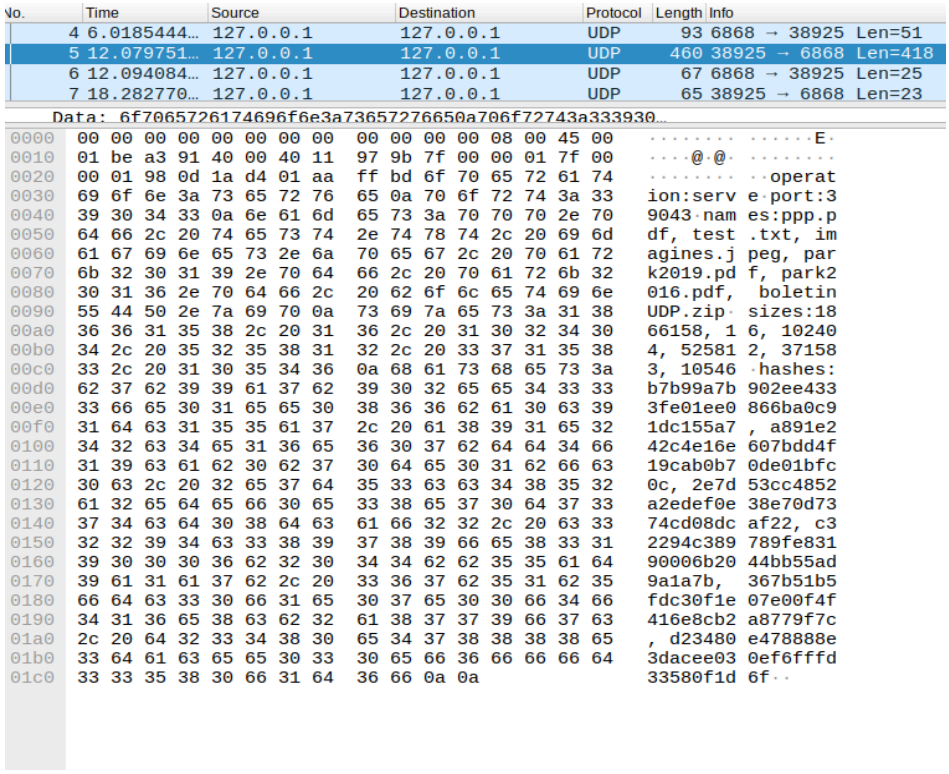


No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000…	127.0.0.1	127.0.0.1	UDP	78	38925 → 6868 Len=36
2	0.0571652…	127.0.0.1	127.0.0.1	UDP	86	6868 → 38925 Len=44
3	6.0145672…	127.0.0.1	127.0.0.1	UDP	65	38925 → 6868 Len=23
4	6.0185444…	127.0.0.1	127.0.0.1	UDP	93	6868 → 38925 Len=51

Frame 2: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface lo, id 0

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  00 48 9d d3 40 00 40 11  9e cf 7f 00 00 01 7f 00   ·H··@·@· ........
0020  00 01 1a d4 98 0d 00 34  fe 47 6f 70 65 72 61 74   .......4 ·Goperat
0030  69 6f 6e 3a 70 69 6e 67  5f 73 75 63 63 65 73 73   ion:ping _success
0040  0a 70 72 6f 74 6f 63 6f  6c 3a 31 32 33 34 35 36   ·protoco l:123456
0050  37 38 39 41 0a 0a                                 789A··
```

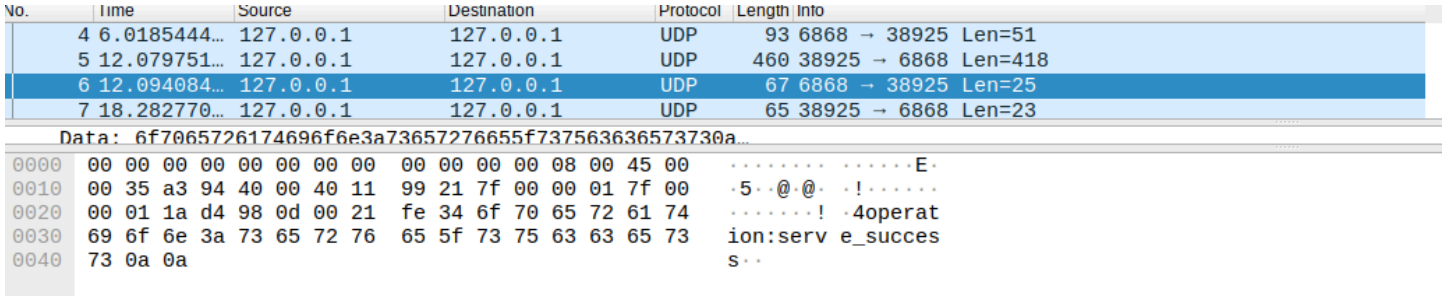
Ejemplo de `serve` y `serve_success`:



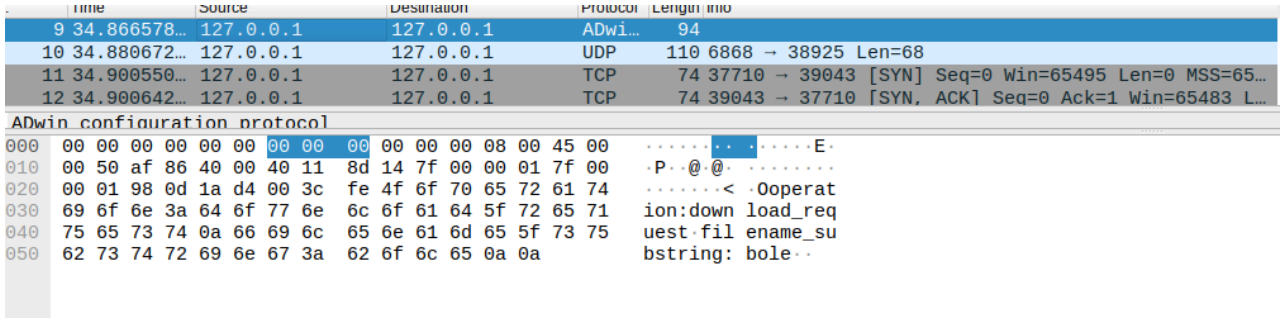
No.	Time	Source	Destination	Protocol	Length	Info
4	6.0185444…	127.0.0.1	127.0.0.1	UDP	93	6868 → 38925 Len=51
5	12.079751…	127.0.0.1	127.0.0.1	UDP	460	38925 → 6868 Len=418
6	12.094084…	127.0.0.1	127.0.0.1	UDP	67	6868 → 38925 Len=25
7	18.282770…	127.0.0.1	127.0.0.1	UDP	65	38925 → 6868 Len=23

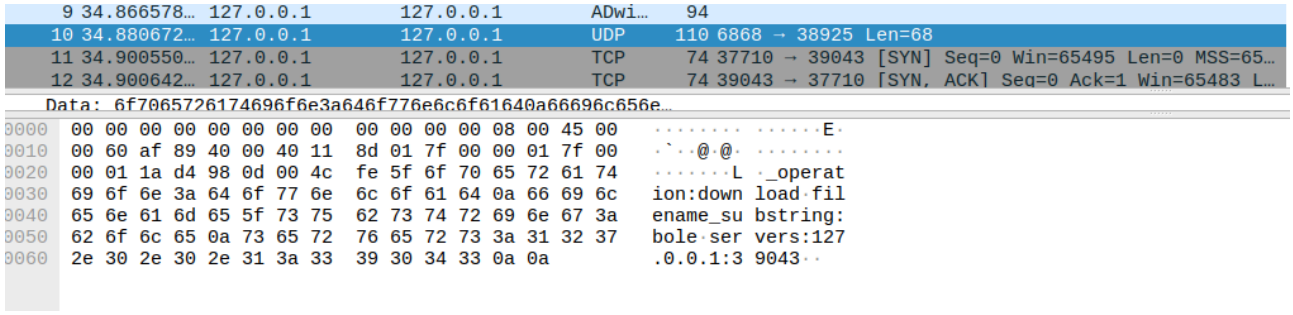
Data: 6f7065726174696f6e3a73657276650a706f72743a333930…

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  01 be a3 91 40 00 40 11  97 9b 7f 00 00 01 7f 00   ····@·@· ........
0020  00 01 98 0d 1a d4 01 aa  ff bd 6f 70 65 72 61 74   ........ ··operat
0030  69 6f 6e 3a 73 65 72 76  65 0a 70 6f 72 74 3a 33   ion:serv e·port:3
0040  39 30 34 33 0a 6e 61 6d  65 73 3a 70 70 70 2e 70   9043·nam es:ppp.p
0050  64 66 2c 20 74 65 73 74  2e 74 78 74 2c 20 69 6d   df, test .txt, im
0060  61 67 69 6e 65 73 2e 6a  70 65 67 2c 20 70 61 72   agines.j peg, par
0070  6b 32 30 31 39 2e 70 64  66 2c 20 70 61 72 6b 32   k2019.pd f, park2
0080  30 31 36 2e 70 64 66 2c  20 62 6f 6c 65 74 69 6e   016.pdf,  boletin
0090  55 44 50 2e 7a 69 70 0a  73 69 7a 65 73 3a 31 38   UDP.zip· sizes:18
00a0  36 36 31 35 38 2c 20 31  36 2c 20 31 30 32 34 30   66158, 1 6, 10240
00b0  34 2c 20 35 32 35 38 31  32 2c 20 33 37 31 35 38   4, 52581 2, 37158
00c0  33 2c 20 31 30 35 34 36  0a 68 61 73 68 65 73 3a   3, 10546 ·hashes:
00d0  62 37 62 39 39 61 37 62  39 30 32 65 65 34 33 33   b7b99a7b 902ee433
00e0  33 66 65 30 31 65 65 30  38 36 36 62 61 30 63 39   3fe01ee0 866ba0c9
00f0  31 64 63 31 35 35 61 37  2c 20 61 38 39 31 65 32   1dc155a7 , a891e2
0100  34 32 63 34 65 31 36 65  36 30 37 62 64 64 34 66   42c4e16e 607bdd4f
0110  31 39 63 61 62 30 62 37  30 64 65 30 31 62 66 63   19cab0b7 0de01bfc
0120  30 63 2c 20 32 65 37 64  35 33 63 63 34 38 35 32   0c, 2e7d 53cc4852
0130  61 32 65 64 65 66 30 65  33 38 65 37 30 64 37 33   a2edef0e 38e70d73
0140  37 34 63 64 30 38 64 63  61 66 32 32 2c 20 63 33   74cd08dc af22, c3
0150  32 32 39 34 63 33 38 39  37 38 39 66 65 38 33 31   2294c389 789fe831
0160  39 30 30 30 36 62 32 30  34 34 62 62 35 35 61 64   90006b20 44bb55ad
0170  39 61 31 61 37 62 2c 20  33 36 37 62 35 31 62 35   9a1a7b,  367b51b5
0180  66 64 63 33 30 66 31 65  30 37 65 30 30 66 34 66   fdc30f1e 07e00f4f
0190  34 31 36 65 38 63 62 32  61 38 37 37 39 66 37 63   416e8cb2 a8779f7c
01a0  2c 20 64 32 33 34 38 30  65 34 37 38 38 38 38 65   , d23480 e478888e
01b0  33 64 61 63 65 65 30 33  30 65 66 36 66 66 66 64   3dacee03 0ef6fffd
01c0  33 33 35 38 30 66 31 64  36 66 0a 0a               33580f1d 6f··
```

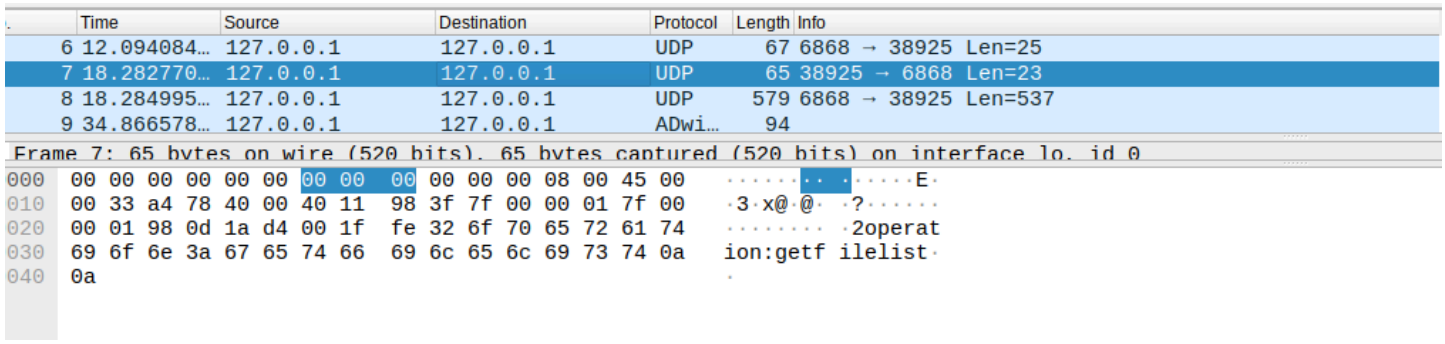


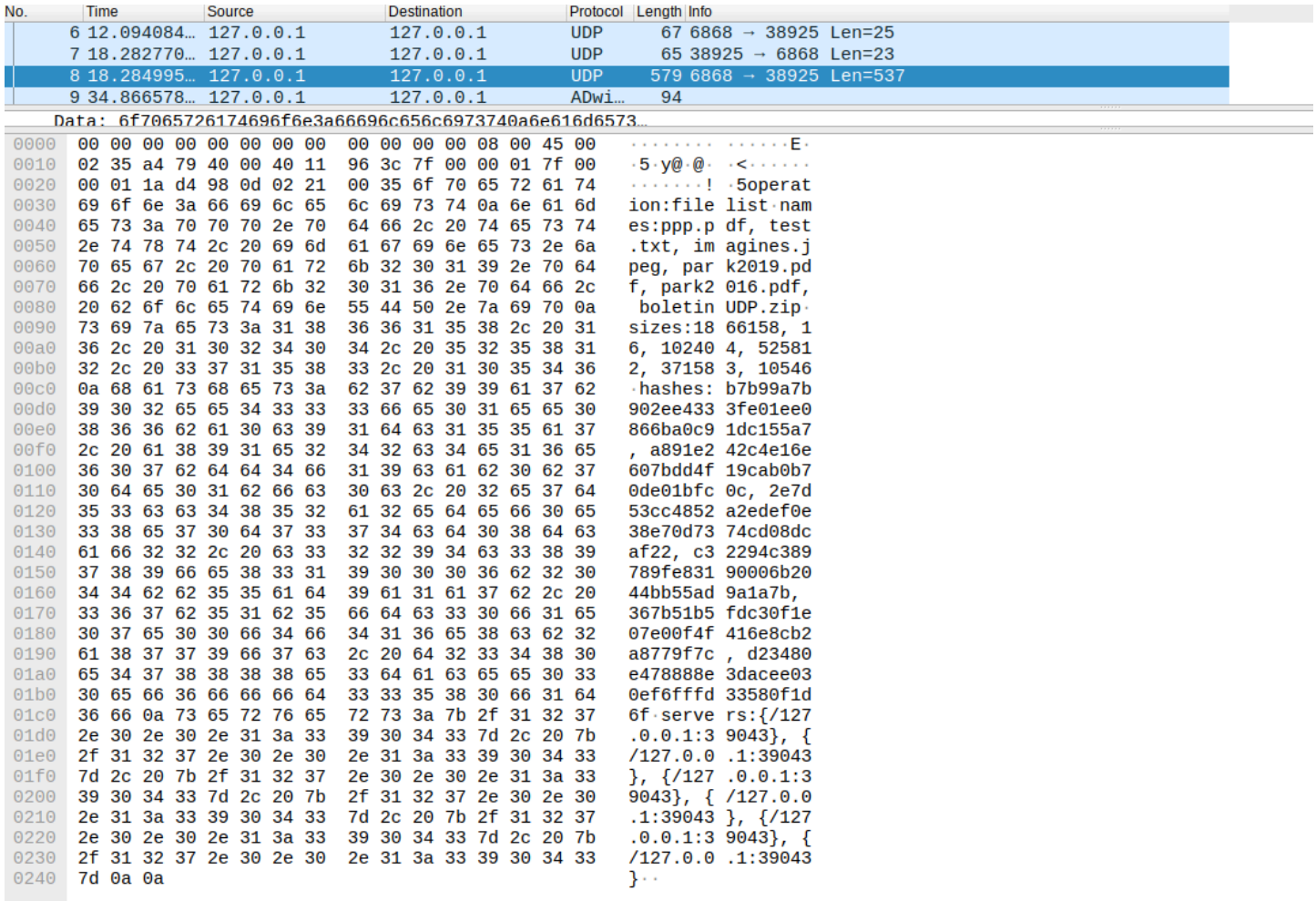
Ejemplo de `download_request` y `download`:





Ejemplo de `getfilelist` y `filelist`:





```
No.  Time        Source     Destination  Protocol Length Info
     6 12.094084… 127.0.0.1  127.0.0.1    UDP      67 6868 → 38925 Len=25
     7 18.282770… 127.0.0.1  127.0.0.1    UDP      65 38925 → 6868 Len=23
     8 18.284995… 127.0.0.1  127.0.0.1    UDP     579 6868 → 38925 Len=537
     9 34.866578… 127.0.0.1  127.0.0.1    ADwi…    94
Data: 6f7065726174696f6e3a66696c656c6973740a6e616d6573…

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  02 35 a4 79 40 00 40 11  96 3c 7f 00 00 01 7f 00   ·5·y@·@· ·<......
0020  00 01 1a d4 98 0d 02 21  00 35 6f 70 65 72 61 74   .......! ·5operat
0030  69 6f 6e 3a 66 69 6c 65  6c 69 73 74 0a 6e 61 6d   ion:file list·nam
0040  65 73 3a 70 70 70 2e 70  64 66 2c 20 74 65 73 74   es:ppp.p df, test
0050  2e 74 78 74 2c 20 69 6d  61 67 69 6e 65 73 2e 6a   .txt, im agines.j
0060  70 65 67 2c 20 70 61 72  6b 32 30 31 39 2e 70 64   peg, par k2019.pd
0070  66 2c 20 70 61 72 6b 32  30 31 36 2e 70 64 66 2c   f, park2 016.pdf,
0080  20 62 6f 6c 65 74 69 6e  55 44 50 2e 7a 69 70 0a    boletin UDP.zip·
0090  73 69 7a 65 73 3a 31 38  36 36 31 35 38 2c 20 31   sizes:18 66158, 1
00a0  36 2c 20 31 30 32 34 30  34 2c 20 35 32 35 38 31   6, 10240 4, 52581
00b0  32 2c 20 33 37 31 35 38  33 2c 20 31 30 35 34 36   2, 37158 3, 10546
00c0  0a 68 61 73 68 65 73 3a  62 37 62 39 39 61 37 62   ·hashes: b7b99a7b
00d0  39 30 32 65 65 34 33 33  33 66 65 30 31 65 65 30   902ee433 3fe01ee0
00e0  38 36 36 62 61 30 63 39  31 64 63 31 35 35 61 37   866ba0c9 1dc155a7
00f0  2c 20 61 38 39 31 65 32  34 32 63 34 65 31 36 65   , a891e2 42c4e16e
0100  36 30 37 62 64 64 34 66  31 39 63 61 62 30 62 37   607bdd4f 19cab0b7
0110  30 64 65 30 31 62 66 63  30 63 2c 20 32 65 37 64   0de01bfc 0c, 2e7d
0120  35 33 63 63 34 38 35 32  61 32 65 64 65 66 30 65   53cc4852 a2edef0e
0130  33 38 65 37 30 64 37 33  37 34 63 64 30 38 64 63   38e70d73 74cd08dc
0140  61 66 32 32 2c 20 63 33  32 32 39 34 63 33 38 39   af22, c3 2294c389
0150  37 38 39 66 65 38 33 31  39 30 30 30 36 62 32 30   789fe831 90006b20
0160  34 34 62 62 35 35 61 64  39 61 31 61 37 62 2c 20   44bb55ad 9a1a7b,
0170  33 36 37 62 35 31 62 35  66 64 63 33 30 66 31 65   367b51b5 fdc30f1e
0180  30 37 65 30 30 66 34 66  34 31 36 65 38 63 62 32   07e00f4f 416e8cb2
0190  61 38 37 37 39 66 37 63  2c 20 64 32 33 34 38 30   a8779f7c , d23480
01a0  65 34 37 38 38 38 38 65  33 64 61 63 65 65 30 33   e478888e 3dacee03
01b0  30 65 66 36 66 66 66 64  33 33 35 38 30 66 31 64   0ef6fffd 33580f1d
01c0  36 66 0a 73 65 72 76 65  72 73 3a 7b 2f 31 32 37   6f·serve rs:{/127
01d0  2e 30 2e 30 2e 31 3a 33  39 30 34 33 7d 2c 20 7b   .0.0.1:3 9043}, {
01e0  2f 31 32 37 2e 30 2e 30  2e 31 3a 33 39 30 34 33   /127.0.0 .1:39043
01f0  7d 2c 20 7b 2f 31 32 37  2e 30 2e 30 2e 31 3a 33   }, {/127 .0.0.1:3
0200  39 30 34 33 7d 2c 20 7b  2f 31 32 37 2e 30 2e 30   9043}, { /127.0.0
0210  2e 31 3a 33 39 30 34 33  7d 2c 20 7b 2f 31 32 37   .1:39043 }, {/127
0220  2e 30 2e 30 2e 31 3a 33  39 30 34 33 7d 2c 20 7b   .0.0.1:3 9043}, {
0230  2f 31 32 37 2e 30 2e 30  2e 31 3a 33 39 30 34 33   /127.0.0 .1:39043
0240  7d 0a 0a                                          }··
```

7.-Conclusiones

Ha sido una de las prácticas más imponentes que hemos tenido, y al principio resultaba abrumador enfrentarse a ella, especialmente por nuestro escaso conocimiento sobre clases como Socket o el propio funcionamiento de un protocolo como UDP. Sin embargo, a medida que íbamos avanzando, aprendiendo de nuestros errores y dedicando tiempo a entender el código, depurarlo y ver cómo se comportaba, empezamos a darnos cuenta de que no era tan complicado como parecía.

Poco a poco, fuimos familiarizándonos con la lógica detrás de cada parte, ganando seguridad y soltura. Definitivamente, ha sido una práctica muy enriquecedora que nos ha ayudado a conectar mejor con muchos conceptos teóricos. Al final, practicar ha sido lo que nos ha permitido aprender de verdad.

Volviendo a ver la práctica de cara a la entrega de la convocatoria de junio/julio, podemos confirmar que ha resultado bastante entretenido seguir con el proyecto. El hecho de tener que implementar lo que antes eran ampliaciones como parte de la funcionalidad obligatoria nos frustra por ver que no eran tan difíciles de implementar y que de haber aprovechado mejor el tiempo, no habríamos tenido problema alguno con la práctica. Sin embargo, nos ha gustado el tener que utilizar nuestro ingenio para poder hacer algunas mejoras sin la guía de los *TODOs*, dejándonos bastante satisfechos con el trabajo que hemos empleado para hacer este proyecto lo más completo posible.