dISEÑO NANOFILES

Miguel Lucas Serrano, Miguel Salinas Egea. Subgrupo 1.3

Índice

1. **Introducción. (2)**
2. **Formato de los mensajes del protocolo de comunicación con el Directorio. (2-5)**
3. **Formato de los mensajes del protocolo de comunicación entre cliente y servidor de ficheros. (5-7)**
4. **Autómatas de protocolo. (8)**
5. **Ejemplo(s) de intercambio de mensajes. (8-10)**
6. **Captura de paquetes con Wireshark (12-15)**
7. Implementación de mejoras (15-17)
8. **Introducción.**

En este documento se especifica el diseño de los protocolos de comunicación con el directorio con sus tipos de mensaje y formatos a la hora de la comunicación.

1. **Formato de los mensajes del protocolo de comunicación con el Directorio**

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

**NOTA: Un mismo formato de mensaje puede ser utilizado por varios tipos de mensajes diferentes (opcodes diferentes)**

Formatos de mensajes

Formato: Control

|  |
| --- |
| **Opcode**  **(1 byte)** |
|  |

Formato: OneParameter

|  |  |
| --- | --- |
| **Opcode**  **(1 byte)** | **Parámetro**  **(4 bytes)** |
|  |  |

Formato: TLV

|  |  |  |
| --- | --- | --- |
| Opcode  (1 byte) | Len  (4 bytes) | Campo  (n bytes) |
|  |  |  |

Formato: TLV\_Var

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Opcode  (1 byte) | Num\_campos  (4 bytes) | Len 1  (4 bytes) | Campo 1  (n1 bytes) | … | Len k  (4 bytes) | Campo  (nk bytes) |
|  |  |  |  | … |  |  |

Formato: Fichero\_Var

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Opcode | Lon\_nombre1 | Nombre1 | Lon\_hash1 | Hash1 | Meta1 |
|  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| …… | Lon\_nombre K | Nombre K | Lon\_hash k | Hash k | Meta k |
| ….. |  |  |  |  |  |

Tipos y descripción de los mensajes

Mensaje: **Login (opcode = 1)**

Formato: Control

Sentido de la comunicación: Cliente→ Directorio

Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para solicitar “iniciar sesión” y obtener el número de servidores que hay disponibles en ese momento. El valor asignado al opcode es 1.

Ejemplo: login localhost

|  |
| --- |
| **Opcode**  **(1 byte)** |
| **1** |

Mensaje: **LoginOk (opcode = 2)**

Formato: OneParameter

Sentido de la comunicación: Directorio → Cliente

Descripción: Este mensaje lo envía el Directorio al cliente para confirmar que el “login” se ha realizado correctamente. El campo “Parámetro” es un entero (4 bytes) que en este caso contiene el número de servidores de ficheros que hay dados de alta en Directorio. El valor asignado al opcode es 2.

Ejemplo: Confirmación de login que indica que hay 0 servidores.

|  |  |
| --- | --- |
| **Opcode**  **(1 byte)** | **Parámetro**  **(4 bytes)** |
| **1** | 0 |

Mensaje: **Register\_User\_Name (opcode = 6)**

Formato: TLV

Sentido de la comunicación: Cliente → Directorio

Descripción: Este mensaje lo envía el cliente al directorio para poder registrar un nickname. El campo Len contiene la longitud del nickname que va a recibir el directorio, que es un entero y en el campo, “campo” se guarda la cadena que contiene el nickname. El valor asignado al opcode es el 6.

Ejemplo: El cliente manda al directorio el comando register con el nombre pepe.

|  |  |  |
| --- | --- | --- |
| Opcode  (1 byte) | Len  (4 bytes) | Campo  (n bytes) |
| 6 | 5 | jpepe (5 bytes) |

Mensaje: **RegisterOk (opcode = 7)**

Formato: Control

Sentido de la comunicación: Directorio → Cliente

Descripción: Este mensaje lo envía el directorio al cliente, informando de que el registro se ha realizado correctamente.

Ejemplo:

|  |
| --- |
| **Opcode**  **(1 byte)** |
| **7** |

Mensaje: **RegisterFail (opcode = 8)**

Formato: Control

Sentido de la comunicación: Directorio → Cliente

Descripción: Este mensaje lo envía el directorio al cliente, informando de que el registro se no se ha podido realizar porque el nick está duplicado.

Ejemplo:

|  |
| --- |
| **Opcode**  **(1 byte)** |
| **7** |

Mensaje: **getUsers(opcode = 12)**

Formato: Control

Sentido de la comunicación: Cliente → Directorio

Descripción: Este mensaje lo envía el cliente al directorio para pedir la lista de usuarios que hay conectados, únicamente tenemos el campo opcode que indica al directorio que tiene que devolver la lista de usuarios.

Ejemplo:

|  |
| --- |
| **Opcode**  **(1 byte)** |
| **12** |

Mensaje: **Userlist** **(opcode = 13)**

Formato: TLV\_VAR

Sentido de la comunicación: Directorio → Cliente

Descripción: Este mensaje lo envía el directorio al cliente creando una lista con los usuarios que están conectados al servidor. El primer campo indica el opcode, el segundo campo “Num\_campos” indica el número de usuarios que están conectados, para los cuales tendremos que guardar la longitud de su nickname en el campo “Len” y el nickname en sí se guardará en el siguiente campo. Habrá tantos campos, como nicknames, para ello utilizamos el campo “Num\_campos”.

Ejemplo: Cuando están conectados los usuarios manuela y ana.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Opcode  (1 byte) | Num\_campos  (4 bytes) | Len 1  (4 bytes) | Campo 1  (n1 bytes) | Len 2  (4 bytes) | Campo 2  (n2 bytes) |
| 13 | 2 | 7 | manuela | 3 | ana |

Mensaje: **ServeFiles** (**opcode = 9**)

Formato: Fichero

Sentido de la comunicación: Cliente → Directorio

Descripción: Se utiliza para que el cliente pueda convertirse en un servidor, y compartir con sus archivos para poder ser descargados por otro cliente.

Ejermplo: El cliente manda al directorio la lista de ficheros que va a subir, en el primer campo nos encontramos con el nombre, en el segundo con el hash y los datos que tiene dentro el fichero.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Opcode | Lon\_nombre1 | Nombre1 | Lon\_hash1 | Hash1 | Meta1 |
| 9 | 8 | Pepe.txt | 18 | 87071b18127a076887 | 100 |

Mensaje: **getFiles(opcode = 14)**

Formato: Control

Sentido de la comunicación: Cliente → Directorio

Descripción: Este mensaje lo envía el cliente al directorio para pedir la lista de ficheros que hay subidos, únicamente tenemos el campo opcode que indica al directorio que tiene que devolver la lista de ficheros.

Ejemplo:

|  |
| --- |
| **Opcode**  **(1 byte)** |
| **14** |

Mensaje: **Filelist** **(opcode = 15)**

Formato: TLV\_VAR

Sentido de la comunicación: Directorio → Cliente

Descripción: Este mensaje lo envía el directorio al cliente creando una lista con los ficheros que están subidos al servidor. El primer campo indica el opcode, el segundo campo “Num\_campos” indica el número de ficheros que hay subidos, para los cuales tendremos que guardar la longitud de su nombre en el campo “Len” y el nombre en sí se guardará en el siguiente campo. Habrá tantos campos, como ficheros, para ello utilizamos el campo “Num\_campos”.

Ejemplo: Cuando hay un fichero Manuela.txt.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Opcode  (1 byte) | Num\_campos  (4 bytes) | Len 1  (4 bytes) | Campo 1  (n1 bytes) | Len 2  (4 bytes) | Campo 2  (n2 bytes) | Len 3  (4 bytes) | Campo 3 |
| 15 | 2 | 11 | Manuela.txt | 3 | 200 | 4 | 87071b18127a076887 |

Mensaje: **Lookup Username (opcode = 3)**

Formato: TLV

Sentido de la comunicación: Cliente → Directorio

Descripción: Este mensaje lo envía el cliente al directorio para poder ver si un usuario está en la lista de servidores de la aplicación.

Ejemplo: El cliente manda al directorio el comando browse jpepe.

|  |  |  |
| --- | --- | --- |
| Opcode  (1 byte) | Len  (4 bytes) | Campo  (n bytes) |
| 3 | 5 | jpepe (5 bytes) |

Mensaje: **Lookup Username Found (opcode = 4)**

Formato: TLV

Sentido de la comunicación: Directorio → Cliente

Descripción: Este mensaje lo envía el directorio al cliente para informarle de la ip y el puerto del usuario que había buscado anteriormente para conectarse como cliente a ese servidor.

Ejemplo: El directorio forma el mensaje de la siguiente manera

|  |  |  |
| --- | --- | --- |
| Opcode  (1 byte) | Len  (4 bytes) | Campo  (n bytes) |
| 4 | 14 | 192.168.3.3:10000 |

Mensaje: **Lookup Username Not Found(opcode = 5)**

Formato: Control

Sentido de la comunicación: Directorio → Cliente

Descripción: Este mensaje lo envía el directorio al cliente informando de que no se ha encontrado el servidor, cuyo Nick había indicado al ejecutar el comando browse <nickname>

Ejemplo:

|  |
| --- |
| **Opcode**  **(1 byte)** |
| **5** |

Mensaje: **Logout(opcode = 18)**

Formato: TLV

Sentido de la comunicación: Cliente → Directorio

Descripción: Este mensaje lo envía el cliente al directorio para poder desconectarte como cliente de un servidor.

Ejemplo: El cliente manda al directorio el comando quit

|  |  |  |
| --- | --- | --- |
| Opcode  (1 byte) | Len  (4 bytes) | Campo  (n bytes) |
| 18 | 5 | jpepe (5 bytes) |

Mensaje: **Quit (opcode = 19)**

Formato: Control

Sentido de la comunicación: Directorio → Cliente

Descripción: Este mensaje lo envía el directorio al cliente informándole de que se ha desconectado del servidor.

Ejemplo:

|  |
| --- |
| **Opcode**  **(1 byte)** |
| **19** |

Mensaje: **ServeStop** **(opcode = 16)**

Formato: TLV

Sentido de la comunicación: Cliente → Directorio

Descripción: Este mensaje lo envía el cliente al directorio para dejar de servir ficheros como servidor.

Ejemplo: El cliente al hacer un fgstop, se envía este mensaje con su nickname.

|  |  |  |
| --- | --- | --- |
| Opcode  (1 byte) | Len  (4 bytes) | Campo  (n bytes) |
| 16 | 5 | jpepe (5 bytes) |

Mensaje: **StopOk (opcode = 17)**

Formato: Control

Sentido de la comunicación: Directorio → Cliente

Descripción: Este mensaje lo envía el directorio al cliente informándole de que ya no funciona como servidor y que el comando fgstop se ha realizado correctamente.

Ejemplo:

|  |
| --- |
| **Opcode**  **(1 byte)** |
| **17** |

1. **Formato de los mensajes del protocolo de comunicación entre cliente y servidor de ficheros**

NOTA: Un mismo formato de mensaje puede ser utilizado por varios tipos de mensajes diferentes (opcodes diferentes)

Formatos de mensajes

Formato: FilehashMessage

operation: xxxxxxxx\n

filehash: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\n

\n

Formato: File

operation: xxxxxxxx\n

datos: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\n

seq: xxxxxxxxxxxxxxxxxxx\n

\n

Formato: Operation

operation: xxxxxxxx\n

\n

Formato: Campos

operation: xxxxxxxx\n

name1:xxxxxx\n

size1:xxxxxxxx\n

hash1:xxxxxxxx\n

…

namen:xxxxxxx\n

sizen:xxxxxxxx\n

hashn:xxxxxxxxxx\n

\n

**…**

**Tipos y descripción de los mensajes**

Mensaje: **Download**

Formato: FilehashMessage

Sentido de la comunicación: Cliente→ Servidor de ficheros

Descripción: Este mensaje lo envía el cliente al servidor de ficheros para solicitar la descarga de un fichero cuyo hash viene dado en el campo “filehash”.

Ejemplo:

operation: download\n

filehash: 87071b18127a076887613fa2b38511ec\n

\n

Mensaje: **file**

Formato: FileHashMessage

Sentido de la comunicación: Servidor de ficheros → cliente

Descripción: Este mensaje lo envía el servidor al cliente devolviéndole el fichero que ha pedido descargar con sus datos. El campo secuencia, nos indica el número de mensajes que nos quedan por enviar de un fichero.

Ejemplo:

operation: file\n

data: sdaddfa\n

seq: 123\n

\n

Mensaje: **fileNotFound**

Formato: Operation

Sentido de la comunicación: Servidor de ficheros → cliente

Descripción: Este mensaje lo envía el servidor al cliente indicando que el fichero no ha sido encontrado.

Ejemplo:

operation: fileNotFound\n

\n

Mensaje: **ServedFiles**

Formato: Campos

Sentido de la comunicación: Servidor de ficheros → cliente

Descripción: Este mensaje lo envía el servidor al cliente indicando la lista de ficheros que se pueden descargar de un servidor. Aquí un ejemplo de como sería el mensaje teniendo el servidor de ficheros, dos ficheros: hola.txt y prueba.txt.

operation: SERVEDFILES\n

name:hola.txt\n

size:10000\n

hash:87071b18127a076887613fa2b38511ec\n

name1:prueba.txt\n

size1:1024\n

hash1:81b18127a076887613fa2b38511ec123\n

\n

Mensaje: **close**

Formato: Operation

Sentido de la comunicación: Cliente → Servidor de ficheros

Descripción: Este mensaje lo envía cliente al servidor indicando que va a desconectarse del servidor.

Ejemplo:

operation: close\n

\n

Mensaje: **queryfiles**

Formato: Operation

Sentido de la comunicación: Cliente → Servidor de ficheros

Descripción: Este mensaje lo envía cliente al servidor de ficheros para conocer la lista de ficheros que se encuentran en el servidor para su descarga.

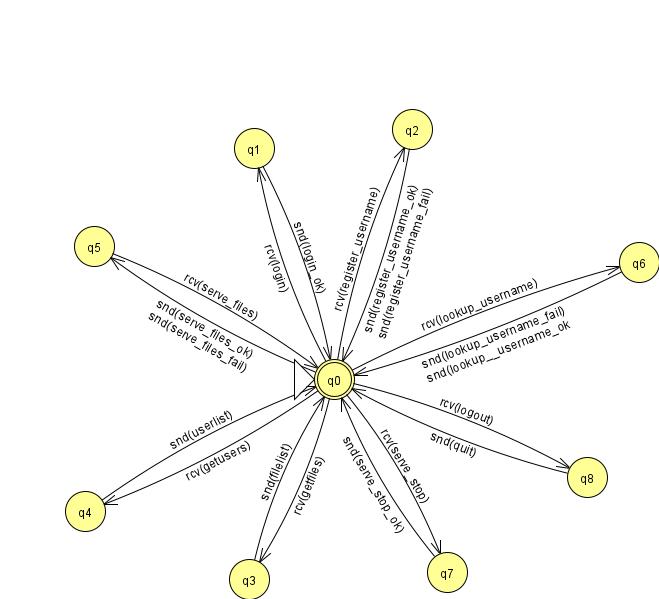
Ejemplo:

operation: queryfiles\n

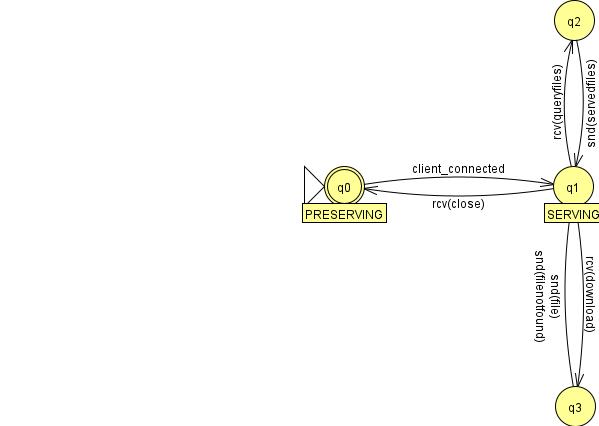
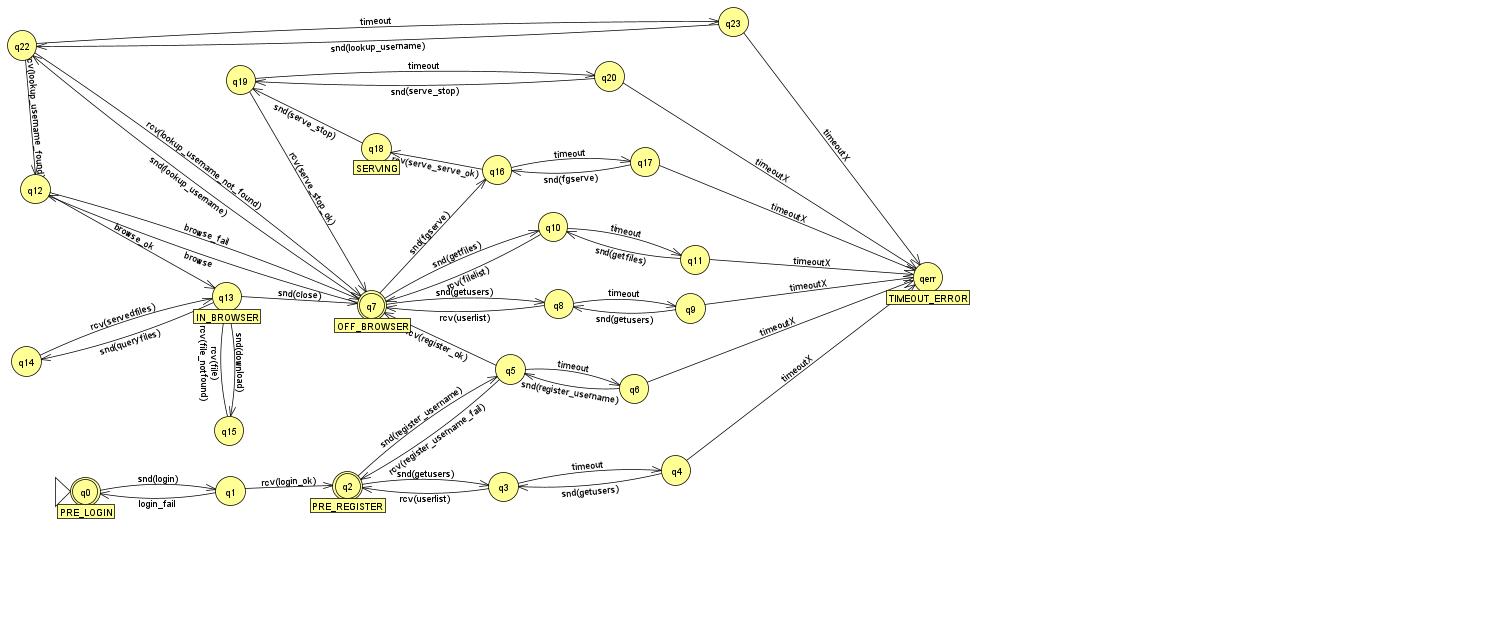
\n

1. **Autómatas de protocolo**

Autómata Servidor de Directorio

****

Autómata Cliente (unificado TCP/UDP) y Autómata Servidor

****

1. **Ejemplo(s) de intercambio de mensajes**

Incluir en esta sección ejemplos de “conversaciones” ficticias haciendo uso de los mensajes definidos en las secciones anteriores y comentando cómo el autómata restringe qué mensaje(s) puede enviar recibir cada extremo de la comunicación en cada instante de la conversación (estado del autómata).

login localhost

(en este momento de la conversación solo se puede hacer login debido a que el autómata restringe que se pueda pasar hasta el siguiente estado si no recibe un login\_ok)

|  |
| --- |
| **Opcode**  **(1 byte)** |
| **1** |

Cuando recibe el login\_ok, tiene varios caminos, puede pedir una lista de usuarios. Aquí se encuentra en el estado Pre-register. Si manda aquí el userlist llegará el mensaje con una lista vacía

userlist(en la línea de comandos)

|  |
| --- |
| **Opcode**  **(1 byte)** |
| **12** |

O registrarse en la aplicación mediante el comando:

register jpepe

|  |  |  |
| --- | --- | --- |
| Opcode  (1 byte) | Len  (4 bytes) | Campo  (n bytes) |
| 6 | 5 | jpepe (5 bytes) |

Se envía un mensaje register username del cliente al directorio. Y si el directorio envía un mensaje register\_ok pasaremos al siguiente estado OFF\_BROWSER. Recibiría este mensaje.

|  |
| --- |
| **Opcode**  **(1 byte)** |
| **7** |

Ahora para poder pasar a servidor de ficheros tenemos que ejecutar el comando fgserve <puerto>, este mensaje se codifica de la siguiente manera:

fgserve 10000

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Opcode | Lon\_nombre1 | Nombre1 | Lon\_hash1 | Hash1 | Meta1 |
| 9 | 8 | Pepe.txt | 18 | 87071b18127a076887 | 100 |

Ahora ejecutaremos otro nanofiles en el que conectaremos un cliente y conectaremos al servidor creado en el anterior comando (suponemos que login y register han funcionado, con nombre paco).

browse <nickname> o <ip>:<puerto>

browse jpepe

Este comando mandaría un mensaje del cliente al servidor de ficheros para conectarse al servidor con ese nombre, se codificaría el mensaje look\_up\_username siguiente:

|  |  |  |
| --- | --- | --- |
| Opcode  (1 byte) | Len  (4 bytes) | Campo  (n bytes) |
| 3 | 5 | jpepe (5 bytes) |

Este mensaje va desde el cliente al directorio, comprobando que ese usuario es ahora un servidor de ficheros. Si está, el directorio devuelve un look\_up\_username\_ok, sino está devuelve un look\_up\_username\_not\_found, mensajes con formato control que únicamente indican el opcode en el mensaje.

Ahora que estamos dentro del servidor podríamos ejecutar un queryfiles para conocer que ficheros están subidos en este servidor:

operation: queryfiles\n

\n

Aparecería el fichero Pepe.txt que es el único que hay subido

Esto aparecería si cuando salgamos, ejecutaramos un filelist.

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode  (1 byte) | Num\_campos  (4 bytes) | Len 1  (4 bytes) | Campo 1  (n1 bytes) |
| 16 | 2 | 4 | Pepe.txt |

En este momento, podemos recibir mensajes de download, si el fichero no se encuentra enviaremos un mensaje del tipo fileNotFound,

operation: download\n

filehash: 87071b18127a076887613fa2b38511ec\n

\n

si lo hemos encontrado enviaremos un mensaje de tipo file con el archivo pedido por el cliente.

operation: file\n

data: Holaepep\n

seq: 1\n

\n

En todos estos estados cabría la posibilidad de cerrar el servidor mediante el mensaje fgstop, que nos enviará al estado OFF\_BROWSER. Esto se haría con el siguiente mensaje serve\_stop:

|  |  |  |
| --- | --- | --- |
| Opcode  (1 byte) | Len  (4 bytes) | Campo  (n bytes) |
| 16 | 5 | jpepe (5 bytes) |

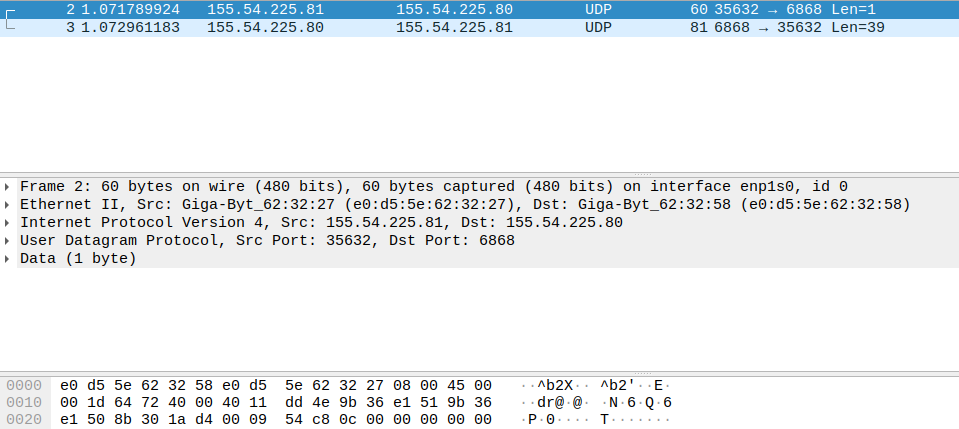
De la misma manera el cliente se puede desconectar del servidor con el comando close que se codifica con el siguiente mensaje desde el cliente al servidor de ficheros.

operation: close\n

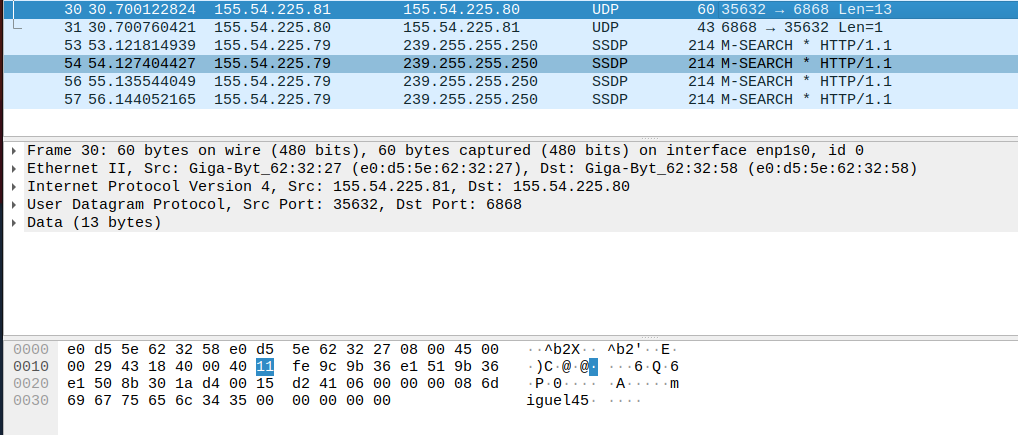
\n

1. Captura de paquetes con Wireshark

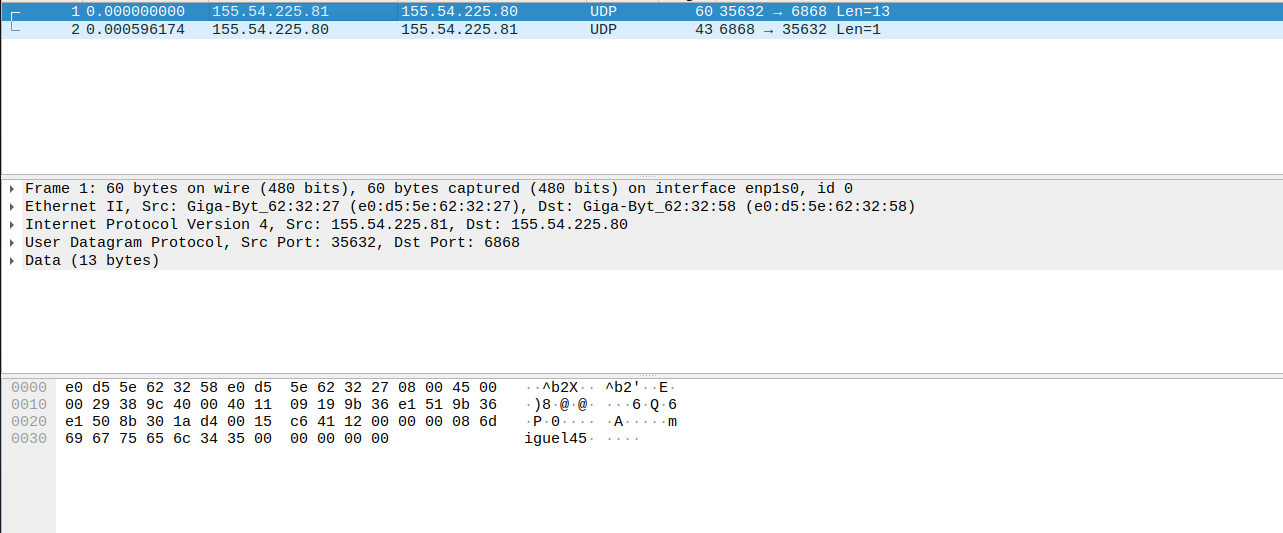
**Userlist:**



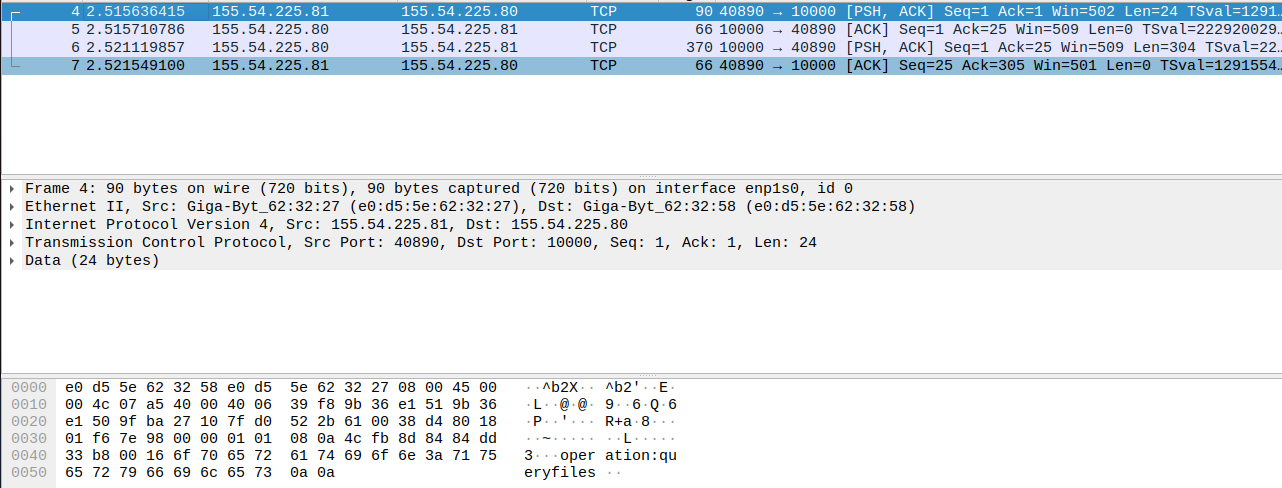
**Register:**



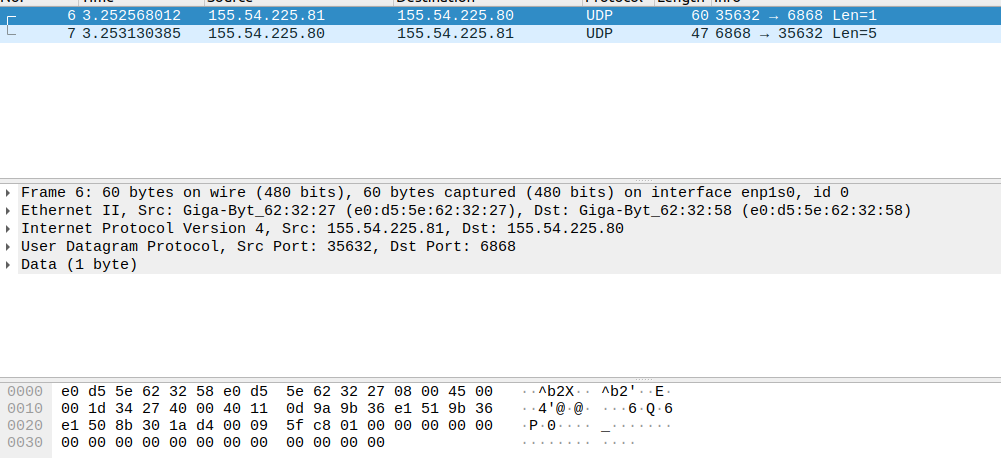
**Quit:**



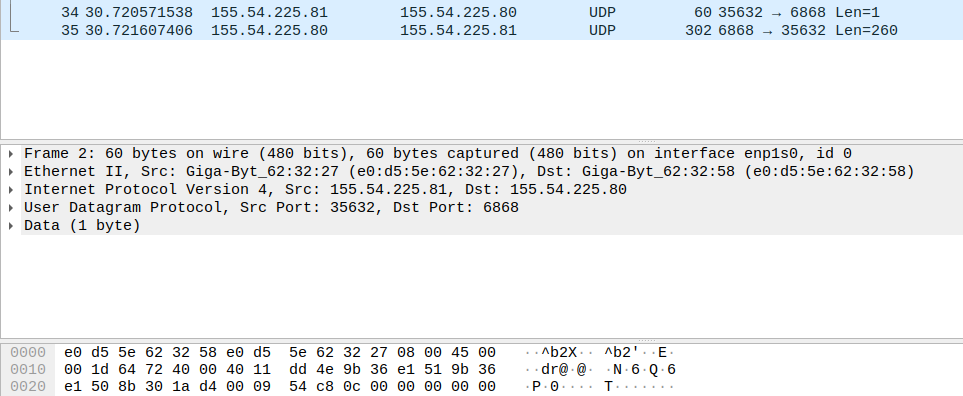
**Queryfiles:**



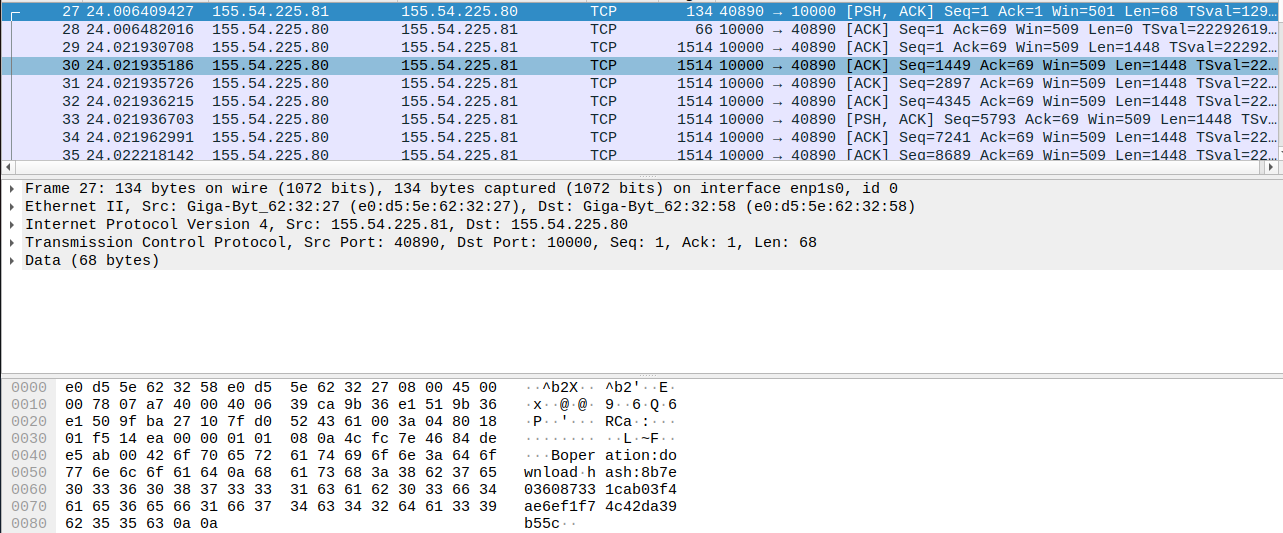
**Login:**



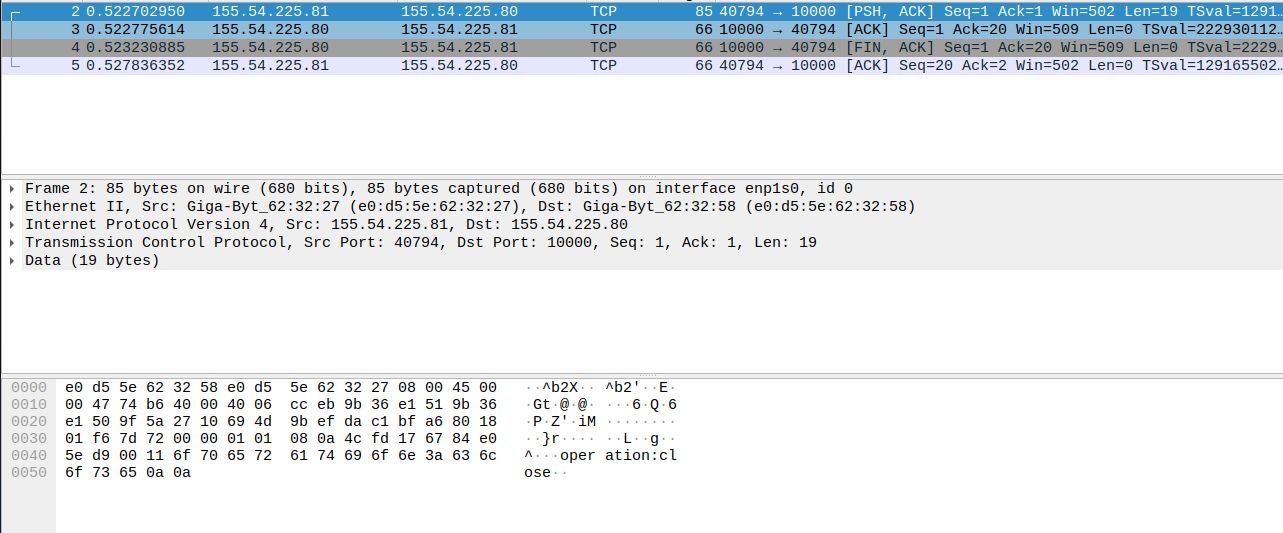
**Filelist:**



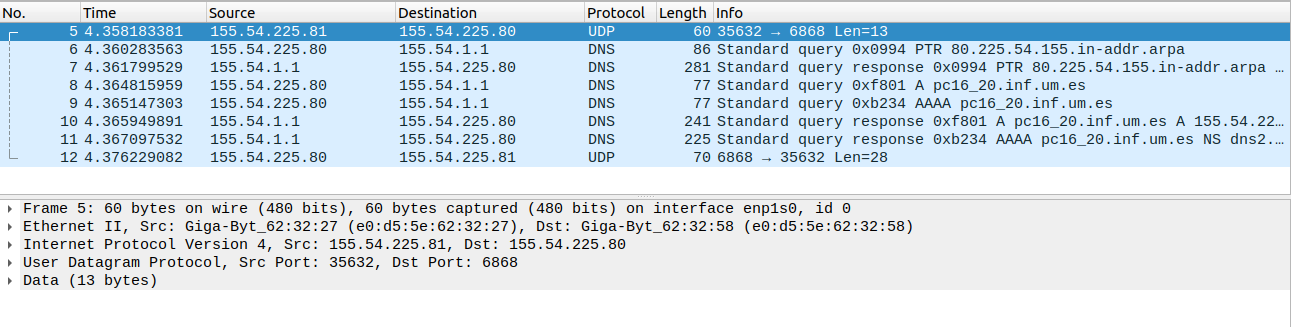
**Download:**



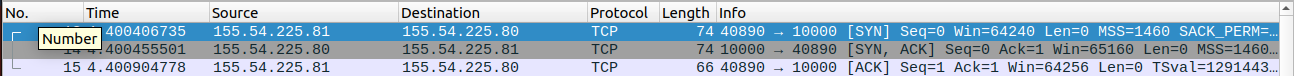
**Close:**



**BrowseUDP:**



**BrowseTCP:**



7.Implementación de mejoras

***FILELIST***

Cuando se recibe el comando filelist, el camino que sigue es el siguiente: Nanofiles reconoce el comando y en la clase NFController comprueba que el estado del cliente en el autómata es el correcto para poder recibir la lista del fichero en el directorio. Esto sucede cuando la variable clientStatus == OFF\_BROWSER. Mediante un objeto controllerDir, vamos a pedir con la función getFileListFromDirectory() la lista de ficheros. Ahora nos movemos a la clase NFControllerLogicDir, aquí es donde realmente vamos a mostrar los ficheros que se encuentran ofrecidos:

**void** getFileListFromDirectory() {

FileInfo[] listaFicheros;

**try** {

listaFicheros = directoryConnector.getFiles();

**if**(listaFicheros.length==0) {

System.***out***.println("No hay ficheros disponibles");

}**else** {

System.***out***.println("Ficheros: ");

FileInfo.*printToSysout*(listaFicheros);

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

Como podemos ver en el código mostrado, a través de un objeto directoryConnector conseguiremos la lista de ficheros del directorio. Dentro de la clase, está función, creará el mensaje para construir la lista de ficheros, para ello hace lo siguiente:

**public** FileInfo[] getFiles() **throws** IOException{

**byte**[] requestData = DirMessage.*buildFileListRequestMessage*();

**byte**[] responseData = **this**.sendAndReceiveDatagrams(requestData);

**return** DirMessage.*processFileListResponseMessage*(responseData);

}

A través de la clase DirMessage construye un mensaje para pedirle al directorio que le proporcione la lista de ficheros a través del mensaje definido anteriormente GETFILES. Ahora con los datos que se han recibido del directorio, se procesa la respuesta y con la clase DirMessage crea el mensaje para devolverle la lista de ficheros (utilizando el formato file). Ahora ya tendríamos la lista de ficheros en el array de FileInfo creado y podremos ver si la lista de ficheros está vacía o llena.

***QUERYFILES***

**protected** **void** browserQueryFiles() {

**try** {

List<FileInfo> listaFicheros = nfConnector.searchFilesQuery();

FileInfo[] ficheros = **new** FileInfo[listaFicheros.size()];

**int** i = 0;

**for** (FileInfo fileInfo : listaFicheros) {

ficheros[i] = fileInfo;

i++;

}

FileInfo.*printToSysout*(ficheros);

} **catch** (IOException e) {

e.printStackTrace();

}

}

Cuando recibimos un comando queryfiles se realiza lo siguiente, a través de un objeto NFConnector nos vamos a comunicar con el servidor. Con la función searchFilesQuery vamos a mandarle al servidor una petición de queryfiles. El servidor, procesará este mensaje y nos enviará un mensaje codificado con la lista de ficheros que hay ahora mismo en este servidor. Codificando el mensaje con uno de tipo SERVEFILES. En el socket dis, se encontrará la lista de ficheros con el nombre, el tamaño y el hash.

**public** List<FileInfo> searchFilesQuery() **throws** IOException{

/\*

\* Le mandamos al servidor el mensaje con la peticion queryfiles

\* y nos llega un mensaje con la lista de ficheros del servidor.

\*/

PeerMessage mensaje = **new** PeerMessage(PeerMessageOps.***OP\_QUERYFILES***);

String codificado = mensaje.toEncodedString();

dos.writeUTF(codificado);

PeerMessage mensajeServidor;

String datosServidor;

datosServidor = dis.readUTF();

mensajeServidor = PeerMessage.*fromString*(datosServidor);

**return** mensajeServidor.getFicheros();

}

***USERLIST(SERVIDOR/USUARIO)***

En la función sendUserList de DirectoryThread comprobamos que los nicks estén en el mapa de servidores, en ese caso les añadimos una cadena que los identifica como servidores.

***BROWSE NICKNAME***

Para buscar un servidor de ficheros por su nombre, distinguimos primero de si el mensaje cuando hacemos el comando browse contiene “:” si contiene, entonces significa que el cliente quiere buscar a través de ip:puerto, sino tenemos que buscar a través del nickname. Para buscar con el nickname, tendremos que comprobar en que socket está escribiendo, para ello mediante un objeto controllerDir y el método lookupUserInDirectory buscamos en que socket está sirviendo este nickname. Este mensaje, a través del Nick pediría al directorio que busque al servidor con ese nickname, si lo encuentra (***OPCODE\_LOOKUP\_USERNAME\_FOUND***) mandaría la InetSocketAddress, si no lo encuentra recibiríamos un null.

***INFORMACIÓN ACTUALIZADA***

En la clase directoryThread aparte del mapa de ficheros que se están sirviendo, tenemos un mapa que consta de los campos hash y nickname, de manera que cuando hacemos el fgserve aparte de

añadir al mapa de ficheros el fichero, también lo añadimos en este mapa auxiliar, mediante el hash del fichero y el nick del propietario del fichero.

Y lo único que hacemos es que cuando se ejecuta la orden fgstop, borramos el nick del mapa de servidores, y tenemos un Hashset, en el que metemos todos los hash del mapa auxiliar.

Luego con un for each comprobamos en el mapa auxiliar que el valor de cada clave hash, es decir el nick asociado a ese hash sea igual que el del nick del objeto dirMessage y en ese momento lo borramos del mapa de ficheros servidos el fichero asociado a ese hash.

***FGSTOP***

Cuando ya no tenemos ningún cliente, al saltar el “timeout” leemos por la entrada estándar para comprobar si el servidor quiere dejar de ser servidor, para volver a ser cliente. Esto sucede en la clase NFServerSimple. Además como ya hemos indicado antes, cuando cerramos el servidor lo quitamos de la lista de servidores.