INDICE

1. Introducción.

En este trabajo hemos diseñado y programado protocolos en Java, nos han servido para poder hacer un sistema de compartición y transferencia de ficheros. Este se basa en dos programas principales, Directory, el cual será el servidor directorio de nuestro trabajo y un conjunto de peers que denominaremos como Nanofiles.
El intercambio o comunicación entre estas dos clases se hace de dos formas, una de ellas es que cada peer se quiera comunicar con el directorio, aquí se va a regir por el modelo cliente-servidor, como hemos visto en teoría, un peer actuará como cliente y el directorio actuará como servidor. Por otro lado, tenemos el modelo de comunicación entre dos peers que es peer-to-peer(P2P), los cuales pueden funcionar tanto como servidor como como cliente.
En este documento también vamos a poder ver los mensajes de los protocolos, a nivel de aplicación, que hemos diseñado para el intercambio de mensajes entre un servidor y un cliente UDP, además vamos a poder ver cuáles son los mensajes entre los peers para poder descargarse ficheros de ambos lados, también se han añadido los autómatas de cada uno de ellos.
Algunas de las restricciones que hemos puesto han sido que cada un cierto de timeout-n se termina el programa dando un error. Y que un Cliente no puede hacer nada a no ser que haga un login, menos fgserve, y le devuelva LoginOk, como que la confirmación de que este login ha salido con éxito y no un fracaso.

2. Protocolos Diseñados

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).

2.1 Formato de los mensajes del protocolo de comunicación con el Directorio

Tipos y descripción de los mensajes

Mensaje: Login
Sentido: CLIENTE->SERVIDOR
-Mensaje para darse de alta en el servidor teniendo de información el nombre de usuario para poder registrarlo.
operation:login\n
nickname:alejandro\n
\n
Sentido: SERVIDOR->CLIENTE
-Mensaje que indica que el inicio de sesión ha sido un éxito o ha fracasado
Operation: login\n
Sessionkey:125\n
loginok:true\n
\n

Mensaje: logout
Sentido: CLIENTE->SERVIDOR
-Mensaje para darse de baja en el servidor.
operation:logout\n
\n
Sentido: SERVIDOR->CLIENTE
-Mensaje que indica una baja de sesión exitosa SERVIDOR->CLIENTE
Operation:logout\n
Logout: true\n
\n

Mensaje: Userlist
Sentido: CLIENTE->SERVIDOR
-Mensaje que pide la lista de usuarios que hay en el servidor y cuales de ellos son servidores de ficheros
operation:UserList\n
\n
Sentido: SERVIDOR->CLIENTE
-Mensaje te devuelde la lista de los usuarios que hay registrados dentro del servidor y lista de los usuarios que son servidores de ficheros
Operation:UserList\n
Userlist: [Alejandro]\n
fileServers: []\n

En el caso en el que alguna de las dos listas esté vacía se imprimirá así:
*Userlist: alumno
*Users que son servidores:

Mensaje:FileList
Sentido: CLIENTE->SERVIDOR
-Mensaje que hace el cliente para poder pedir la lista de los ficheros que han subido lo peers al directorio con los metadatos
operation:FileList \n
\n
Sentido: SERVIDOR->CLIENTE
-mensaje que muestra los ficheros que han compartido los otros peers que han sido publicados en el directorio
Operation: FileList \n
Filelist: Ajaaa, 480 bytes, cfsjaiadue78wrw7qd7a8daud8ad;
\n
en el caso en el que no haya no podría nada

Mensaje: Publish
Sentido: CLIENTE->SERVIDOR
-Mensaje que indica que va a publicar los metadatos de los ficheros de su carpeta compartida, en nuentro caso será nf-shared
operation:Publish\n
\n

Sentido: SERVIDOR -> CLIENTE
-Mensaje que te confirma la exitosa o no entrega de los archivos subidos.
Operation: publish_response\n
Publishresponse: true\n
\n

Mensaje: Search
Sentido: CLIENTE->SERVIDOR
-Mensaje que indica el <filehash> donde quiere buscar la lista
operation:search\n
search:<fileHash> \n
\n

Sentido: SERVIDOR -> CLIENTE
-Mensaje que indica la lista de los nicknames de los servidores de ficheros que lo tienen disponible (y lo han publicado al directorio con publish) que le devuelve el servidor
listofnicknames:ListOfNicknames\n
\n

Un ejemplo de ListOfNicknames seria: nickname1, nickname2, nickname3, ...
en el caso de no tener ninguno no pondría nada

Mensaje: StopServer
Sentido: CLIENTE->SERVIDOR
-Mensaje para dar de baja a un usuario que se ha dado de alta como servidor de ficheros anteriormente con bgserve
operation:unregister_server \n
\n
Sentido: SERVIDOR -> CLIENTE
-Mensaje que indica que se ha dado de baja correctamente, o no, a un servidor de ficheros
Operation: unregisterOk\n
\n

Mensaje: Bgserve
Sentido: CLIENTE->SERVIDOR
-Mensaje que indica que un usuario se quiere dar de alta como servidor de ficheros
operation:Register_file_server\n
\n

Sentido: SERVIDOR -> CLIENTE
-Mensaje que indica que este usuario se ha podido dar de baja correctamente como servidor de ficheros
Operation: registerok\n
\n

Mensaje: IP_request | Port
Sentido: CLIENTE->SERVIDOR
-Mensaje que pide el puerto y la Ip del nickname que se le ha pasado como parámetro al downloadfrom para poder estableces una conexión con él y poder descargarte los ficheros
operation:Reques_Ip | Port \n
\n
Sentido: SERVIDOR -> CLIENTE
-Mensaje que te devuelve el servidor (SERVIDOR -> CLIENTE)
Operation: Request_ip\n
Request_ip: /127.0.0.1\n
Port: 57732\n
\n

2.1 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.

Tipos y descripción de los mensajes
EJEMPLO:
Mensaje:**FileNotFound (opcode = 1)**
Sentido de la comunicación:Servidor de ficheros → Cliente
Descripción:Este mensaje lo envía el par servidor de ficheros al par 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)
1

Mensaje: Donwload_from
Sentido: SERVIDOR -> CLIENTE
-Mensaje con los archivos que se han sido descargados

Opcode (1 byte)	Longitud	Valor
2	8 bytes	n bytes

Mensaje: Download_ok
Sentido: SERVIDOR -> CLIENTE
-mensaje que te indica que la descarga se ha realizado con éxito

Opcode (1 byte)	Longitud	Valor
1	8 bytes	n bytes

Mensaje: DOWNLOAD FAIL
Sentido: SERVIDOR -> CLIENTE
-Mensaje que te indica que la descarga no se ha podido realizar con éxito

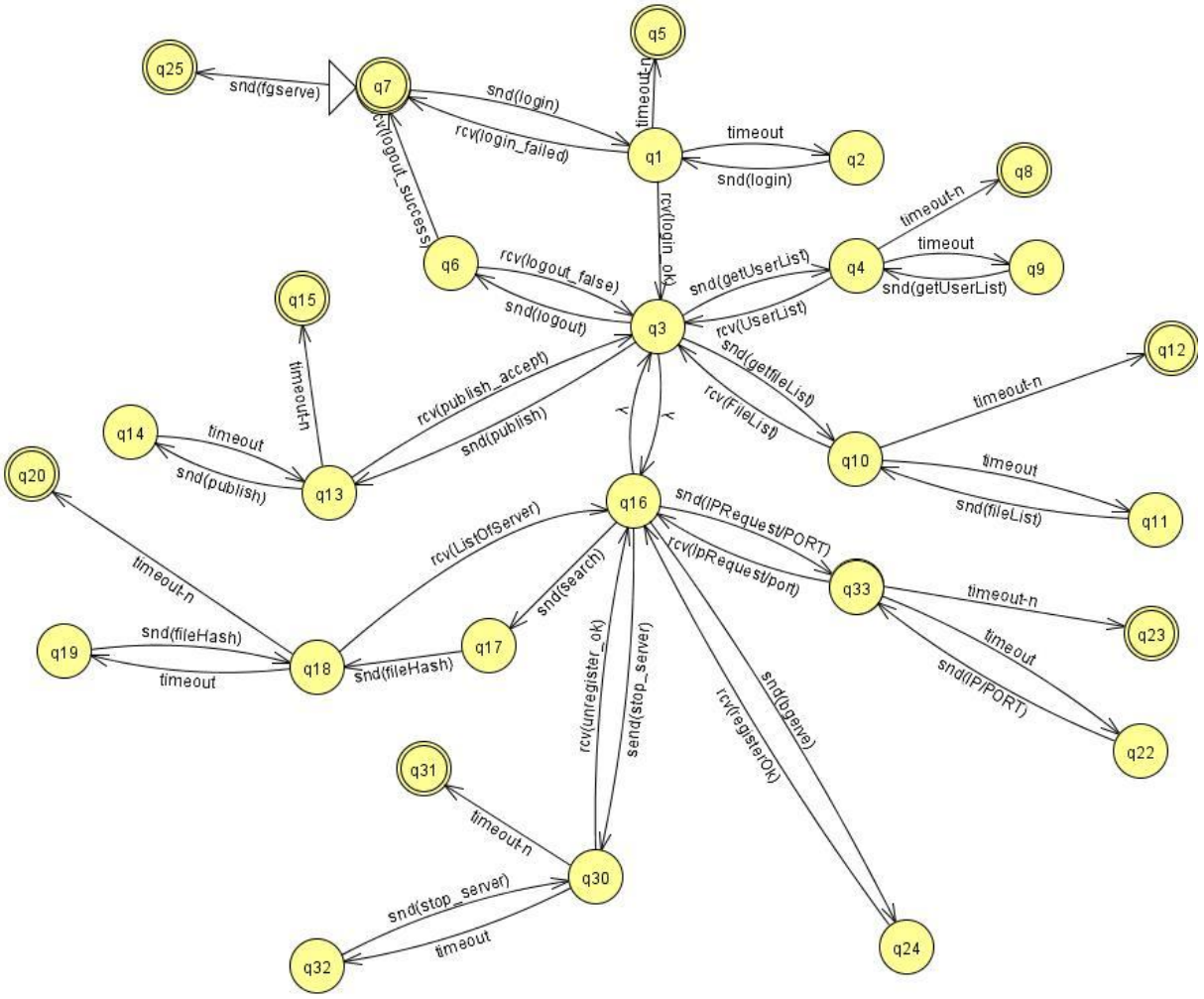
Opcode (1 byte)
3

2.2 Autómatas de protocolo

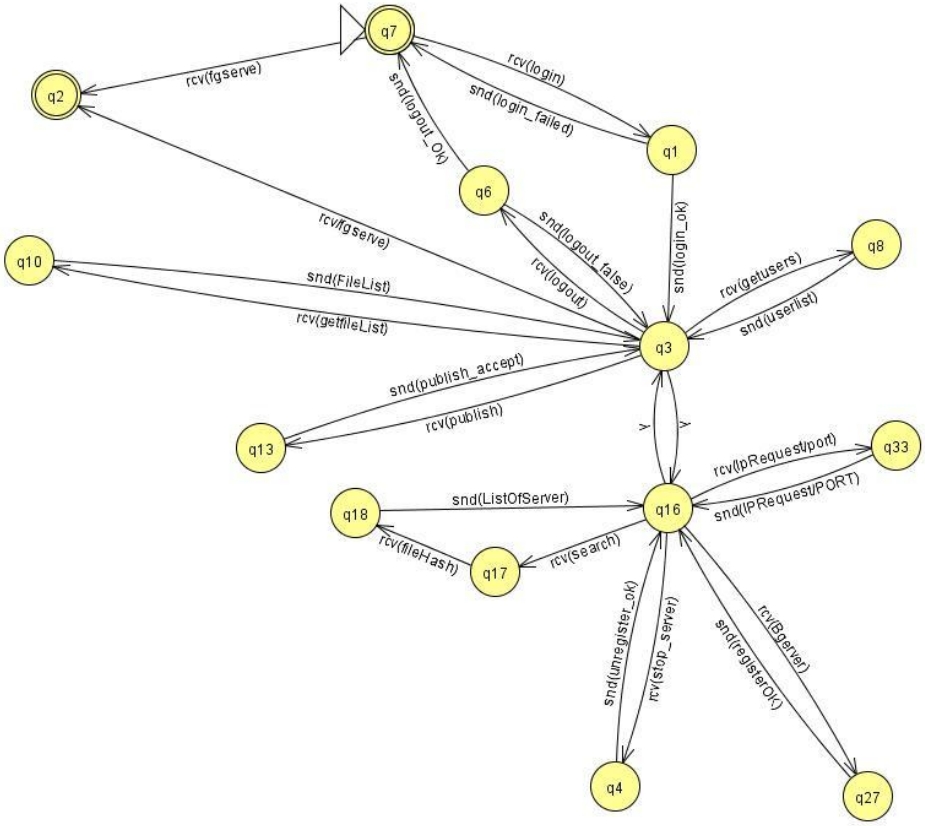
Con respecto a los autómatas, hemos considerado las siguientes restricciones:

- Un cliente del directorio no puede pedir la lista de usuarios, ni hacer algo que no sea fgserve o quit si no ha iniciado sesión previamente.
- Cada vez que no reciba nada, va a pasar un tiempo (timeout) y se lo va a volver a enviar
- Cada cierto número de timeouts el programa va a terminar debido a un error
- También hemos puesto unos estados de lambda para que del login pueda hacer cada una de las operation y que el autómata no se quedara tan apretado
- Cuando pone snd es enviar y cuando pone rcv es recibir.

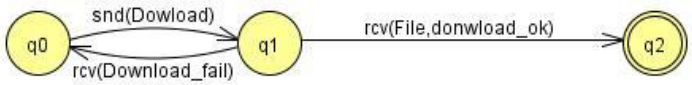
Autómata rol cliente de directorio

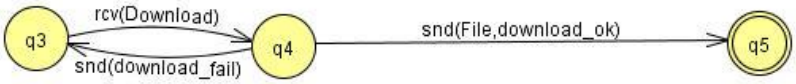


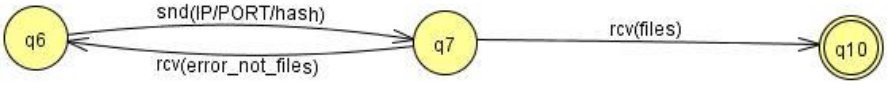
Autómata rol servidor de directorio

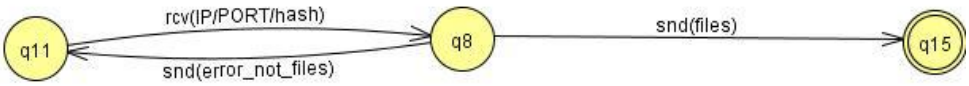


Autómata rol cliente de ficheros y Autómata rol servidor de ficheros









2.4. Ejemplo de intercambio de mensajes

Incluir en esta sección ejemplos de “conversaciones” ficticias (con valores inventados) 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).

//UN CLIENTE APARTE DEL PRINCIPAL SE CONECTA A UN SERVIDOR EN PRIMER PLANO PARA PODER DESCARGARNOS COSAS DE ÉL EN UN FUTURO (Funcionará como DIRECTORIO en el ejemplo principal)

CLIENTE-SERVIDOR: *hace fgserve sin login, para ser un servidor en primer plano(no le llega nada al directorio principal)*
fgserve
```
*Servidor en el puerto: 10000
```

CLIENTE: *Cliente principal inicia sesión*
login localhost alumno
```
*SessionKey:4767
*logged in Ok
```

DIRECTORIO : *Recibe el inicio de sesión*
```
operation:login
sessionkey:4767
loginok:true
```

CLIENTE: *Intenta iniciar sesión otra vez de manera fallida*
login locahost borja
```
* You cannot login because you are not logged out from the directory
```

CLIENTE: *Pide la lista de usuarios*
userlist
```
*Encontrando a los usuarios:
*UserList: alumno
* Users que son servidores:
```

DIRECTORIO: *Recibe la pedida de usuarios*
```
operation: UserList
userlist: [alumno]
fileservers:[]
```

CLIENTE: *Publica los ficheros de la carpeta compartida*
publish
```
*Successfully published!
```

DIRECTORIO: *Recibe la petición de publicar los ficheros, y la ejecuta*
```
operation:Publish_response
publishresponse:true
```

CLIENTE: *Pide la lista de archivos*
filelist
```
ajaaaaaaaa 53 41ef7184b6fe68907501e05b671ef3275c4c52ed
cosas.txt 478 7ec1d43011dfda049d4550b5343f47149f479136;
```

DIRECTORIO: *Envía la lista de ficheros*
```
operation:FileList
filelist:ajaaaaaaaa, 53 bytes, 41ef7184b6fe68907501e05b671ef3275c4c52ed; cosas.txt, 478
bytes, 7ec1d43011dfda049d4550b5343f47149f479136
```

CLIENTE: *Muestra los archivos que tenemos actualmente*
myfiles
```
List of files in local folder:
Name                                Size Hash
ajaaaaaaaa                            53 41ef7184b6fe68907501e05b671ef3275c4c52ed
cosas.txt                            478 7ec1d43011dfda049d4550b5343f47149f479136
```

CLIENTE: *Descarga del servidor en primer plano "ajaaaaaaa" y lo guarda como asdjajk.txt*
downloadfrom localhost:10000 41ef7184b6fe asdjajk.txt
```
Conexión TCP establecida.
* File downloaded successfully: asdjajk.txt
```

CLIENTE: *Crea un servidor en segundo plano*
bgserve
```
* BackgroundServer operando en el puerto 64238
*Enviando registro al directorio...
*Servidor conectado correctamente
```

DIRECTORIO: *Recibe la petición del cliente y lo registra como servidor en segundo plano*
```
operation:RegisterOk
registerok:true
```

CLIENTE2: *Inicia sesión con el mismo nombre*
login localhost alumno
```
* Usuario ya registrado
*logIntoDirectory ha devuelto error
```

DIRECTORIO: *Deniega el inicio de sesión*
```
operation:login
loginok:false
```

(Tendríamos que volver a ejecutar NanoFiles.java)
CLIENTE2: *Inicia sesión con otro nombre*
login localhost borja
```
* SessionKey: 5919
*logged in Ok
```

DIRECTORIO: *Acepta el inicio de sesión*

```
operation:login
sessionkey:5919
loginok:true
```

CLIENTE2: *Pide la lista de usuarios*

userlist

```
*Encontrando a los usuarios:
*UserList: borja, alumno
*Users que son servidores: alumno
```

DIRECTORIO: *Muestra la lista de usuarios*

```
operation:UserList
userlist:[borja, alumno]
fileservers:[alumno]
```

CLIENTE2: *Descarga del servidor en segundo plano del cliente1*

downloadfrom alumno 7ec1d43011dfda049d4550b5343f47149f479136 cosas2.txt

```
Conexión TCP establecida.
* File downloaded successfully: cosas2.txt
```

DIRECTORIO: *Pide la ip correspondiente a alumno*

```
* solicitud de Ip
Sending message to client operation:request_ip
request_ip:/127.0.0.1
port:64238
```

CLIENTE: *Para el servidor en segundo plano*

stopserver

```
* Servidor detenido.
*Enviada la solicitud para darse de baja
*Te has dado de baja correctamente
```

DIRECTORIO: *Da de baja el servidor*

```
operation:UnregisterServer
unregisterok:true
```

AMBOS CLIENTES: *Cierran la sesión*

logout

```
*logged out Ok
```

DIRECTORIO:*Confirma el cierre de sesión*

```
operation:logout
logout:true
```

3. Mejoras implementadas y breve descripción de su programación

Las mejoras que hemos implementado son las siguientes:

`fgserve` puerto variable	0,5 punto(s)
`downloadfrom` por *nickname*	1 punto(s)
`bgserve` secuencial	1 punto(s)
`bgserve` multihilo	0,5 punto(s)
`stopserver`	0,5 punto(s)
`bgserve` puerto efímero	0,5 punto(s)
`userlist` ampliado con servidores	0,5 punto(s)
`publish` + `filelist`	0,5 punto(s)

En el código están comentadas, y aquí vamos a hacer una pequeña explicación de cómo está hecha su implementación.
Fgserve puerto variable: lo que hemos hecho aquí es en la clase NFServerSimple, que el puerto predeterminado (10000) se lo hemos puesto a una nueva variable llamada port, cuando creamos el socketAddress si este ve que el puerto 10000 está ocupado, lo que hace es avisar por pantalla que este puerto esta ocupado y le suma uno a la variable y lo vuelve a intentar.

Downloadfrom por nickname: Aquí hemos añadido dos operaciones nuevas en DirMessage las cuales son, PORT y REQUEST_IP lo que hacen es buscar la ip y el puerto del nickname que se le ha pasado como parámetro, hemos implementado en NFControllerLogicDir en la función getServerAddress (a la cual llama NFController), la función que llama lookupServerAddrByUsername, la cual llamaría a directoryConnetor. lookupServerAddrByUsername con el nickname que le hemos pasado y dentro de aquí ya conseguiría la Ip y el puerto que necesitamos

Bgserve: aquí lo que hemos hecho ha sido, primero comprobar que no hay ningún otro servidor Bgserve, una vez que hemos hecho esto creamos un nuevo NFServer y le hacemos un. startServer, esta función se encuentra en NFServer y te crea un nuevo hilo de la siguiente manera new Thread(this). start();
Dentro del NFServer lo que hacemos es crear (como siempre hacemos) un nuevo InetSocketAddress con puerto 0 y creamos un nuevo serverSocket. Mas tarde damos de alta al usuario como servidor de ficheros. Creamos la operación de Register_file_server.

stopServer: llamado por NFController, con el servidor que tenemos creado, llamamos a. stopserver, este lo que hace es poner a true la variable stopServer y cierra el serverSocket, luego da de baja al servidor a través de unregisterServer en la clase NFControllerServer que lo que hace es crear un DirMessage con el nickname y la sessionkey para borrarlo del array de filesServers. Creamos una nueva operación llamada Unregister_server, además de comprobar que exista algún servidor de ficheros activo.

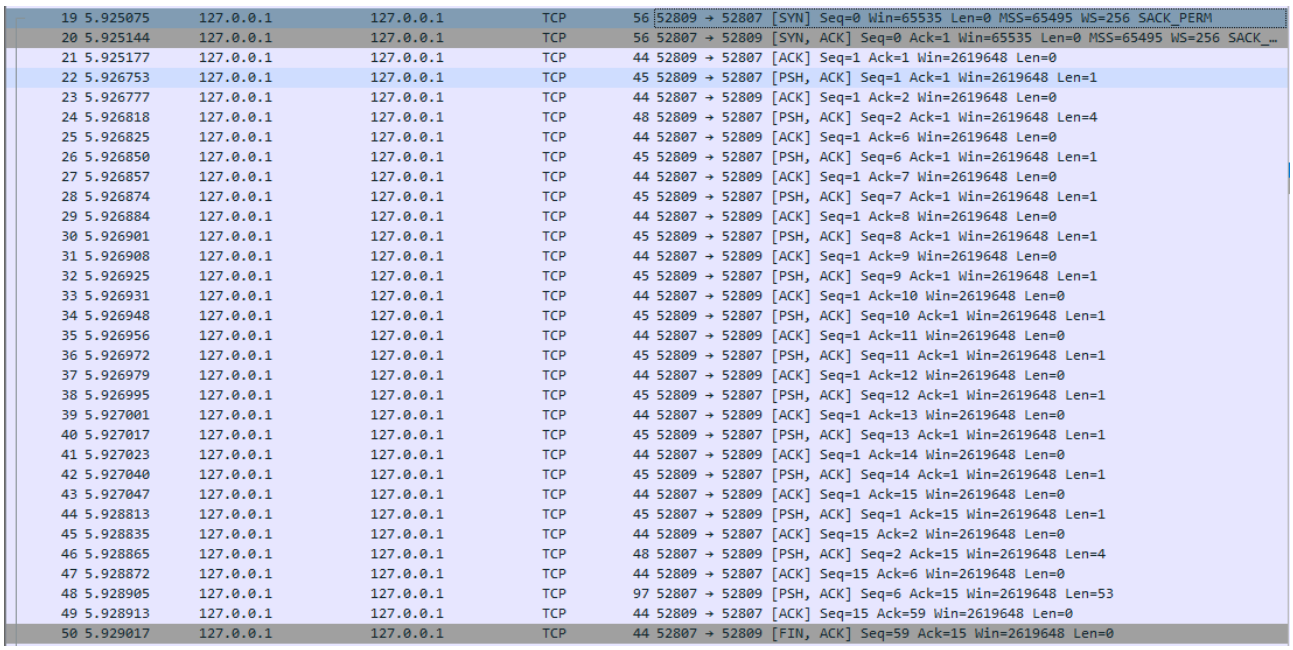
Bgserve puerto efímero: cuando creamos el socketAddress en vez de ponerle un puerto en particular le ponemos un 0 y nos cogerá alguno aleatorio y que este libre.

Userlist con servidores de ficheros: cuando hacemos bgserve damos de alta a un usuario como servidor de ficheros, y lo añadimos a un array, una vez que tenemos esto hemos cambiado un poco la función de getUserList para que en vez de un array nos devuelva un mapa con los arrays de los usuarios y con el array de los filesServer, cada uno de estos con una key diferente: “users” y “fileservers”. Dentro de getAndPrintUserList cogemos los respectivos arrays y los mostramos por pantalla.

Publish + fileList:
Primero lo que hacemos es publish, el cual coge todos los ficheros de nuestra carpeta compartida, y llama a la función publishLocalFiles en directoryConnector, este crea el archivo para publicar los archivos y lo envía, recibe una respuesta del servidor y verifica que la publicación a sido un éxito. Hemos creado dos nuevas operaciones llamada publish y publish_response.
FileList lo que hacemos es en un string que tenemos con ficheros que nos ha dejado el publish, los cogemos y los ponemos por pantalla

4. Capturas de WireShark

En el bgserve lo que se hace es lo siguiente, en la captura podemos ver uno con más longitud que otra, en este caso es de 53, ahí es donde está el archivo que le hemos pedido al peer, primero lo que hace es empezar una conversación (lo gris), después sigue con esta conversación mandándole mensajes para poder pedirle el fichero, se lo manda y ya cierran la conversación.



19	5.925075	127.0.0.1	127.0.0.1	TCP	56	52809 → 52807 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
20	5.925144	127.0.0.1	127.0.0.1	TCP	56	52807 → 52809 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_…
21	5.925177	127.0.0.1	127.0.0.1	TCP	44	52809 → 52807 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
22	5.926753	127.0.0.1	127.0.0.1	TCP	45	52809 → 52807 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=1
23	5.926777	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [ACK] Seq=1 Ack=2 Win=2619648 Len=0
24	5.926818	127.0.0.1	127.0.0.1	TCP	48	52809 → 52807 [PSH, ACK] Seq=2 Ack=1 Win=2619648 Len=4
25	5.926825	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [ACK] Seq=1 Ack=6 Win=2619648 Len=0
26	5.926850	127.0.0.1	127.0.0.1	TCP	45	52809 → 52807 [PSH, ACK] Seq=6 Ack=1 Win=2619648 Len=1
27	5.926857	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [ACK] Seq=1 Ack=7 Win=2619648 Len=0
28	5.926874	127.0.0.1	127.0.0.1	TCP	45	52809 → 52807 [PSH, ACK] Seq=7 Ack=1 Win=2619648 Len=1
29	5.926884	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [ACK] Seq=1 Ack=8 Win=2619648 Len=0
30	5.926901	127.0.0.1	127.0.0.1	TCP	45	52809 → 52807 [PSH, ACK] Seq=8 Ack=1 Win=2619648 Len=1
31	5.926908	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [ACK] Seq=1 Ack=9 Win=2619648 Len=0
32	5.926925	127.0.0.1	127.0.0.1	TCP	45	52809 → 52807 [PSH, ACK] Seq=9 Ack=1 Win=2619648 Len=1
33	5.926931	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [ACK] Seq=1 Ack=10 Win=2619648 Len=0
34	5.926948	127.0.0.1	127.0.0.1	TCP	45	52809 → 52807 [PSH, ACK] Seq=10 Ack=1 Win=2619648 Len=1
35	5.926956	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [ACK] Seq=1 Ack=11 Win=2619648 Len=0
36	5.926972	127.0.0.1	127.0.0.1	TCP	45	52809 → 52807 [PSH, ACK] Seq=11 Ack=1 Win=2619648 Len=1
37	5.926979	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [ACK] Seq=1 Ack=12 Win=2619648 Len=0
38	5.926995	127.0.0.1	127.0.0.1	TCP	45	52809 → 52807 [PSH, ACK] Seq=12 Ack=1 Win=2619648 Len=1
39	5.927001	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [ACK] Seq=1 Ack=13 Win=2619648 Len=0
40	5.927017	127.0.0.1	127.0.0.1	TCP	45	52809 → 52807 [PSH, ACK] Seq=13 Ack=1 Win=2619648 Len=1
41	5.927023	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [ACK] Seq=1 Ack=14 Win=2619648 Len=0
42	5.927040	127.0.0.1	127.0.0.1	TCP	45	52809 → 52807 [PSH, ACK] Seq=14 Ack=1 Win=2619648 Len=1
43	5.927047	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [ACK] Seq=1 Ack=15 Win=2619648 Len=0
44	5.928813	127.0.0.1	127.0.0.1	TCP	45	52807 → 52809 [PSH, ACK] Seq=1 Ack=15 Win=2619648 Len=1
45	5.928835	127.0.0.1	127.0.0.1	TCP	44	52809 → 52807 [ACK] Seq=15 Ack=2 Win=2619648 Len=0
46	5.928865	127.0.0.1	127.0.0.1	TCP	48	52807 → 52809 [PSH, ACK] Seq=2 Ack=15 Win=2619648 Len=4
47	5.928872	127.0.0.1	127.0.0.1	TCP	44	52809 → 52807 [ACK] Seq=15 Ack=6 Win=2619648 Len=0
48	5.928905	127.0.0.1	127.0.0.1	TCP	97	52807 → 52809 [PSH, ACK] Seq=6 Ack=15 Win=2619648 Len=53
49	5.928913	127.0.0.1	127.0.0.1	TCP	44	52809 → 52807 [ACK] Seq=15 Ack=59 Win=2619648 Len=0
50	5.929017	127.0.0.1	127.0.0.1	TCP	44	52807 → 52809 [FIN, ACK] Seq=59 Ack=15 Win=2619648 Len=0

Wireshark · Seguir secuencia TCP (tcp.stream eq 2) · Adapter for loopback traffic capture

```
....(7ec1d43011dfda049d4550b5343f47149f479136.....el nfcontroller solo tenemos que poner lo del
automata
el nfshell no tocar
nfcontroller logicdir logica necesaria para que haga lo de los comandos, es donde tenemos que ha
cer las cosas
junto el directory connector y el nfdirectory server
nos falta poner que si tenemos ya el nombre logueado, no podemos hacer otro login con ese nombre
nfp2p las acciones con las descargas de ficheros, nf connector se conecta con un servidos aqui s
e descarga y el nfserverComm los envia
```

Algunos ejemplos de otras instrucciones, en este caso son UDP los mensajes.

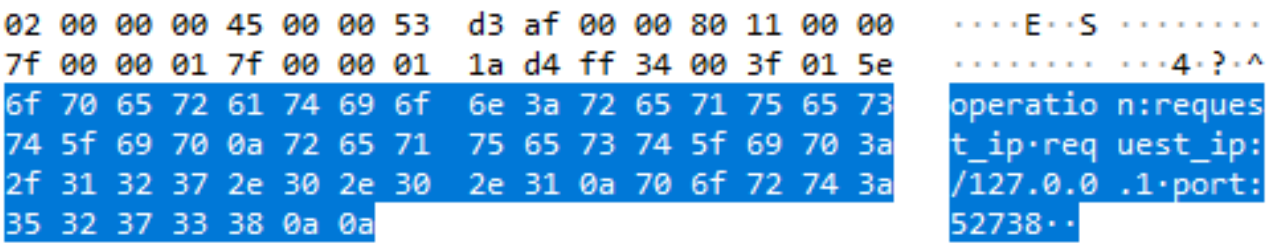
15	9.565664977	155.54.223.229	155.54.223.230	UDP	75 56516 → 6868 Len=33
16	9.589036480	155.54.223.230	155.54.223.229	UDP	88 6868 → 56516 Len=46
93	83.664426199	155.54.223.229	155.54.223.230	UDP	94 56516 → 6868 Len=52
94	83.666969795	155.54.223.230	155.54.223.229	UDP	95 6868 → 56516 Len=53
127	113.269175608	155.54.223.229	155.54.223.230	UDP	107 56516 → 6868 Len=65
128	113.271323816	155.54.223.230	155.54.223.229	UDP	80 6868 → 56516 Len=38
213	182.071797382	155.54.223.229	155.54.223.230	UDP	73 52626 → 6868 Len=31
214	182.072887771	155.54.223.230	155.54.223.229	UDP	88 6868 → 52626 Len=46
258	225.592287503	155.54.223.229	155.54.223.230	UDP	80 52626 → 6868 Len=38
259	225.594551281	155.54.223.230	155.54.223.229	UDP	102 6868 → 52626 Len=60
275	237.049464803	155.54.223.229	155.54.223.230	UDP	61 52626 → 6868 Len=19
276	237.236981480	155.54.223.230	155.54.223.229	UDP	91 6868 → 52626 Len=49
287	247.127455698	155.54.223.229	155.54.223.230	UDP	62 52626 → 6868 Len=20
288	247.128919806	155.54.223.230	155.54.223.229	UDP	193 6868 → 52626 Len=151

```
0000  2c 56 dc fc 46 44 2c 56  dc fc 46 49 08 00 45 00   ,V··FD,V ··FI··E·
0010  00 3d 48 2a 40 00 40 11  fc 4c 9b 36 df e5 9b 36   ·=H*@·@· ·L·6···6
0020  df e6 dc c4 1a d4 00 29  f6 73 6f 70 65 72 61 74   ·······) ·soperat
0030  69 6f 6e 3a 6c 6f 67 69  6e 0a 6e 69 63 6b 6e 61   ion:logi n·nickna
0040  6d 65 3a 61 6c 75 6d 6e  6f 0a 0a                  me:alumn o··
```

```
0000  2c 56 dc fc 46 44 2c 56  dc fc 46 49 08 00 45 00   ,V··FD,V ··FI··E·
0010  00 50 77 fb 40 00 40 11  cc 68 9b 36 df e5 9b 36   ·Pw·@·@· ·h·6···6
0020  df e6 dc c4 1a d4 00 3c  f6 86 6f 70 65 72 61 74   ·······< ··operat
0030  69 6f 6e 3a 55 73 65 72  4c 69 73 74 0a 6e 69 63   ion:User List·nic
0040  6b 6e 61 6d 65 3a 61 6c  75 6d 6e 6f 0a 73 65 73   kname:al umno·ses
0050  73 69 6f 6e 6b 65 79 3a  31 39 32 30 0a 0a         sionkey: 1920··
```

```
0000  2c 56 dc fc 46 44 2c 56  dc fc 46 49 08 00 45 00   ,V··FD,V ··FI··E·
0010  00 30 d0 bb 40 00 40 11  73 c8 9b 36 df e5 9b 36   ·0··@·@· s··6···6
0020  df e6 cd 92 1a d4 00 1c  f6 66 6f 70 65 72 61 74   ········ ·foperat
0030  69 6f 6e 3a 46 69 6c 65  4c 69 73 74 0a 0a         ion:File List··
```

```
02 00 00 00 45 00 00 5c  d3 55 00 00 80 11 00 00   ····E··\ ·U······
7f 00 00 01 7f 00 00 01  1a d4 ff 34 00 48 c6 53   ········ ···4·H·S
6f 70 65 72 61 74 69 6f  6e 3a 55 73 65 72 4c 69   operatio n:UserLi
73 74 0a 75 73 65 72 6c  69 73 74 3a 5b 61 6c 6c   st·userl ist:[all
6c 6c 6c 2c 20 61 6c 6f  5d 0a 66 69 6c 65 73 65   lll, alo ]·filese
72 76 65 72 73 3a 5b 61  6c 75 6d 6e 6f 5d 0a 0a   rvers:[a lumno]··
```



5. CONCLUSIONES

Este ha sido uno de los trabajos más difíciles que vamos a poder tener en toda la carrera, en este trabajo hemos aprendido que los protocolos son más complejos e importantes de lo que de verdad creemos, hemos aprendido mucho sobre el diseño y la implementación de protocolos de comunicación entre un cliente y un Servidor, en este caso en Java, nos ha ayudado a comprender mejor los conceptos como cuales son las dos arquitecturas de cliente—Servidor y de peer-to-peer.

Además, hemos podido fortalecer nuestras habilidades a la hora de implementar y a la hora de usar Eclipse (entorno de trabajo) y nuestra habilidad en el lenguaje usado, ya mencionado, que es Java. Esta práctica nos ha enseñado también la importancia que tiene la buena comunicación entre ambos usuarios ya sea cliente-servidor o peer-peer, pero especialmente esta primera, ya que no nos garantizan la fiabilidad de la comunicación, ya que es UDP. Una de las cosas a las que nos hemos enfrentado también ha sido el como hemos de abordar los problemas que esta práctica nos ha puesto, que no son pocos, y nos ha enseñado a que, aunque todo esté lleno de problemas, con un poco de paciencia y esfuerzo se pueden resolver y mejorarlos.