

# Redes de Comunicaciones :

## Proyecto *Nanofiles*

<b>Autores, ambos del grupo 3.2:</b>
<i>Sánchez Pardo, Jesús</i>
<i>Lin, Jiahui</i>

---

<b>Introducción.....</b>	<b>3</b>
<b>1. Formato de los mensajes del protocolo de comunicación con el Directorio.....</b>	<b>4</b>
<b>2. Formato de los mensajes del protocolo de transferencia de ficheros.....</b>	<b>10</b>
<b>3. Autómatas de protocolo.....</b>	<b>12</b>
<b>4. Ejemplo de intercambio de mensajes.....</b>	<b>15</b>
<b>5. Implementación de las mejoras.....</b>	<b>19</b>
<b>6. Ejemplo Wireshark.....</b>	<b>21</b>
<b>7. Conclusiones.....</b>	<b>26</b>

# Introducción

---

En este documento se especifica el diseño de los protocolos para el funcionamiento del proyecto Nanofiles. Para ello, empezaremos primero con la definición de los protocolos de comunicación tanto cliente-servidor como Peer-To-Peer.

En cuanto a las mejoras implementadas, hemos implementado las siguientes:

Mejora	Puntuación Máxima
Puerto variable para fgserve	0.5
stopserver	0.5
Puerto efímero para bgserve	0.5
publish + filelist	0.5
publish + search	0.5
Baja ficheros y servidores	0.5
download secuencial	1
userlist ampliado con servidores	0.5
filelist ampliado con servidores	0.5

Además, hemos incluido como mejora extra el comando “fgstop”.

# 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 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 y los datos contenidos en estos).

## Tipos y descripción de los mensajes

- Mensaje: **login**

Sentido de la comunicación: Cliente a Directorio

Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para solicitar “iniciar sesión” y registrar el nickname indicado en el mensaje.

Ejemplo:

```
operation: login\n\nnickname: alicia\n\n\n
```

- Mensaje: **login ok**

Sentido de la comunicación : Directorio a Cliente

Descripción: Este mensaje es enviado por el Directorio para notificar al cliente NanoFiles que el login ha sido un éxito, devolviendo además su correspondiente sessionKey (un número aleatorio generado del 1 al 10000, único para él mientras esté conectado). Si el cliente recibe la solicitud, a partir de ahora tendrá que adjuntar en cada mensaje que envíe su sessionKey.

Ejemplo:

```
operation:loginok\n\nsessionkey: [número entre uno y diez mil]\n\n\n
```

- Mensaje: **username\_already\_registered**

Sentido de la comunicación: Directorio a Cliente

Descripción: Mensaje enviado por el directorio cuando el nombre especificado por el usuario ya se encuentra registrado en el directorio.

Ejemplo:

```
operation:username_already_registered\n\n\n
```

- Mensaje: **invalid\_username**

Sentido de la comunicación: Directorio a Cliente

Descripción: Mensaje enviado por el directorio cuando el nombre de usuario indicado por el cliente contiene algún carácter especificado por el directorio como inválido.

Ejemplo:

```
operation:invalid_username\n\n\n
```

---

- Mensaje: **logout**

Sentido de la comunicación: Cliente a Directorio

Descripción: Este mensaje lo envía el cliente para solicitar “cerrar sesión” en el directorio al que lo envía.

Ejemplo:

operation: logout\n

sessionkey: sessionkey del usuario\n

\n

- Mensaje: **logout ok**

Sentido de la comunicación : Directorio a Cliente

Descripción : Este mensaje es enviado cuando se ha podido realizar correctamente el logout solicitado por el cliente.

Ejemplo:

operation: logoutok\n

\n

- Mensaje: **getuserlist**

Sentido de la comunicación : Cliente a Directorio

Descripción : Este mensaje es enviado cuando el cliente solicita ver los usuarios conectados en ese momento al Directorio.

Ejemplo:

operation: getuserlist\n

sessionKey: sessionKey del usuario\n

\n

- Mensaje: **getuserlist ok**

Sentido de la comunicación : Directorio a Cliente

Descripción : Este mensaje es enviado para notificar al cliente que la solicitud ha sido un éxito, además de devolver la lista de usuarios conectados en ese instante. Para indicar si un usuario está registrado como fichero, justo al final de su nickname se añade un asterisco.

**Aclaración:** Cabe aclarar que debido al uso de las comas como separador, el usuario tiene prohibido el uso de comas en su nickname, al igual que el uso de asteriscos.

Ejemplo:

operation: getuserlistok\n

nickname: usuario1,usuario2,usuario3,...,usuari\n

\n

---

- Mensaje: **filelist**

Sentido de la comunicación: Cliente a Directorio

Descripción: Este mensaje se envía cuando el cliente quiere ver los ficheros publicados por otros clientes en el directorio

Ejemplo:

operation: filelist\n

sessionkey: sessionkey del usuario\n

\n

- Mensaje: **filelist\_ok**

Sentido de la comunicación: Directorio a Cliente

Descripción: Mensaje que incluye los ficheros publicados en el directorio.

**Aclaración:** Para separar los servidores que han publicado un mismo fichero, se separan consecutivamente con un asterisco, mientras que para separar listas de servidores de distintos ficheros se ha utilizado una coma.

Ejemplo:

operation: filelist\_ok\n

nickname: (servidores que han publicado el file1), (servidores que han publicado el file2)...(servidores que han publicado el fileN)\n

filenames: file1,file2,file3...fileN\n

\n

- Mensaje: **register\_server**

Sentido de la comunicación: Cliente a Directorio

Descripción: mensaje enviado cuando el cliente quiere convertirse en un servidor de ficheros en segundo plano, por lo que tiene que darse de alta en el directorio indicando el puerto en el que escucha.

Ejemplo:

operation: register\_server\n

sessionkey: sessionkey del cliente\n

port:(puerto entre el 10000 y el 65535)\n

\n

- Mensaje: **server\_registered**

Sentido de la comunicación: Directorio a Cliente

Descripción: mensaje enviado para indicar al cliente que se le ha podido dar de alta.

Ejemplo:

operation: server\_registered\n

\n

- Mensaje: **unregister\_server**

Sentido de la comunicación: Cliente a Directorio

Descripción: mensaje enviado cuando el cliente quiere darse de baja en el directorio como servidor de ficheros.

---

Ejemplo:

operation: register\_server\n  
sessionkey: sessionkey del cliente\n  
\n

- Mensaje: **server\_unregistered**

Sentido de la comunicación: Directorio a Cliente

Descripción: mensaje enviado para indicar al cliente que se le ha podido dar de baja.

Ejemplo:

operation: server\_unregistered\n  
\n

- Mensaje: **invalid\_operation**

Sentido de la comunicación: Directorio a Cliente

Descripción: Mensaje de respuesta a un mensaje del cliente cuyo campo de operación es inválida

Ejemplo:

operation: invalid\_operation\n  
\n

- Mensaje: **register\_server\_fail**

Sentido de la comunicación: Directorio a Cliente

Descripción: mensaje de respuesta para informar que el puerto de escucha del cliente es inválido o está ocupado por otro.

Ejemplo:

operation: register\_server\_fail\n  
\n

- Mensaje: **publish**

Sentido de la comunicación: Cliente a Directorio

Descripción: mensaje enviado para publicar un fichero en el directorio.

Ejemplo:

operation: publish\n  
sessionkey: sessionkey del cliente\n  
\n

- Mensaje: **publish\_ok**

Sentido de la comunicación: Directorio a Cliente

Descripción: mensaje confirmación de la publicación del fichero enviado por el cliente.

---

Ejemplo:

operation: publish\_ok\n  
\n

- Mensaje: **search**

Sentido de la comunicación: Cliente a Directorio

Descripción: mensaje enviado al directorio para localizar un fichero dado un file hash pasado por el cliente.

Ejemplo:

operation: search\n  
sessionkey: sessionkey del cliente\n  
filehashs: file hash pasado por el cliente\n  
\n

- Mensaje: **search\_ok**

Sentido de la comunicación: Directorio a Cliente

Descripción: mensaje enviado para notificar la lista de usuarios que disponen del fichero solicitado.

Ejemplo:

operation: search\_ok\n  
nickname: user 1, user 2, ..., user n\n  
\n

- Mensaje: **ambiguous\_filehash**

Sentido de la comunicación: Directorio a Cliente

Descripción: mensaje enviado para indicar al cliente que el file hash identifica a más de un fichero publicado, es decir, no es único.

Ejemplo:

operation: ambiguous\_filehash\n  
\n

- Mensaje: **file\_not\_found**

Sentido de la comunicación: Directorio a Cliente

Descripción: Mensaje enviado para indicar al cliente que no se ha encontrado ningún fichero publicado con el filehash facilitado por el cliente.

Ejemplo:

operation: file\_not\_found\n  
\n



---

- Mensaje: **get\_serveraddr**

Sentido de la comunicación: Cliente a Directorio

Descripción: Mensaje enviado por el cliente cuando solicita realizar una descarga de un servidor de ficheros conociendo únicamente el nickname de este servidor.

Ejemplo:

```
operation: get_serveraddr\n
sessionkey: sessionkey del cliente\n
nickname: nickname del servidor de ficheros\n
\n
```

- Mensaje : **get\_serveraddr\_ok**

Sentido de la comunicación: Directorio a Cliente

Descripción: Mensaje enviado por el directorio cuando el nombre de usuario indicado por el cliente en la solicitud se encuentra conectado en el directorio y además está como servidor de ficheros.

Ejemplo:

```
operation: get_serveraddr_ok\n
server_address: ip:host del servidor de ficheros\n
\n
```

- Mensaje: **get\_serveraddr\_fail**

Sentido de la comunicación: Directorio a Cliente

Descripción: Mensaje enviado por el directorio cuando el nombre de usuario indicado por el cliente no se encuentra como servidor de ficheros registrado en el directorio.

Ejemplo:

```
operation: get_serveraddr_fail\n
\n
```

## 2. 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

- Mensaje: **Invalid\_Code(opcode = 0)**  
Sentido de la comunicación : Servidor de ficheros → Cliente  
Descripción: Este mensaje se envía para indicar que el código de operación enviado por el cliente es inválida.

Opcode
1

- Mensaje: **DownloadFrom (opcode = 1)**  
Sentido de la comunicación : Cliente → Servidor de Ficheros.  
Descripción: Este mensaje lo envía el cliente al servidor de ficheros cuando desea descargar un fichero identificado por un filehash (o una parte de este).

Opcode	Longitud del filehash	Filehash
1 byte	1 byte	n bytes

- Mensaje: **File (opcode = 2)**  
Sentido de la comunicación: Servidor de Ficheros → Cliente  
Descripción: Mensaje enviado por el servidor de ficheros en caso de que el downloadfrom se haya realizado correctamente (el **hash sólo identifica a un único fichero o a varios ficheros cuya única diferencia es en el nombre del fichero**). Este mensaje contiene el contenido del fichero, la cantidad de bytes que se han leído, y el hash completo del fichero descargado con el fin de que el usuario pueda comprobar si la descarga se ha realizado correctamente (este último hash debe ser igual al hash del fichero destino). Los dos últimos campos son usados cuando se ejecuta el comando download.

Opcode	Cantidad de bytes leído	Contenido del fichero leído	Número de servidores que tienen el fichero	ID del servidor de ficheros
1	8 bytes	n bytes	1 byte	1 byte

- Mensaje: **FileNotFound (opcode = 3)**

Sentido de la comunicación: Servidor de ficheros → Cliente

Descripción: Mensaje que envía el servidor de ficheros cuando la solicitud de descarga del 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: **FileHash Ambiguous(opcode = 4)**

Sentido de la comunicación : Servidor de ficheros → Cliente

Descripción: Este mensaje notifica que el file hash indicado por el cliente identifica a más de un fichero publicado **con diferente contenido** (se pueden identificar a varios ficheros cuya única diferencia sea el nombre de estos)

Opcode (1 byte)
1

- Mensaje: **Download(opcode = 5)**

Sentido de la comunicación : Cliente → Servidor de ficheros

Descripción: Este mensaje es enviado por el cliente a un servidor de ficheros para descargar parte parte de un fichero (**un chunk**). Parecido a DownloadFrom, se incluyen dos nuevos campos para poder realizar la correcta descarga y correcta escritura en el fichero destino.

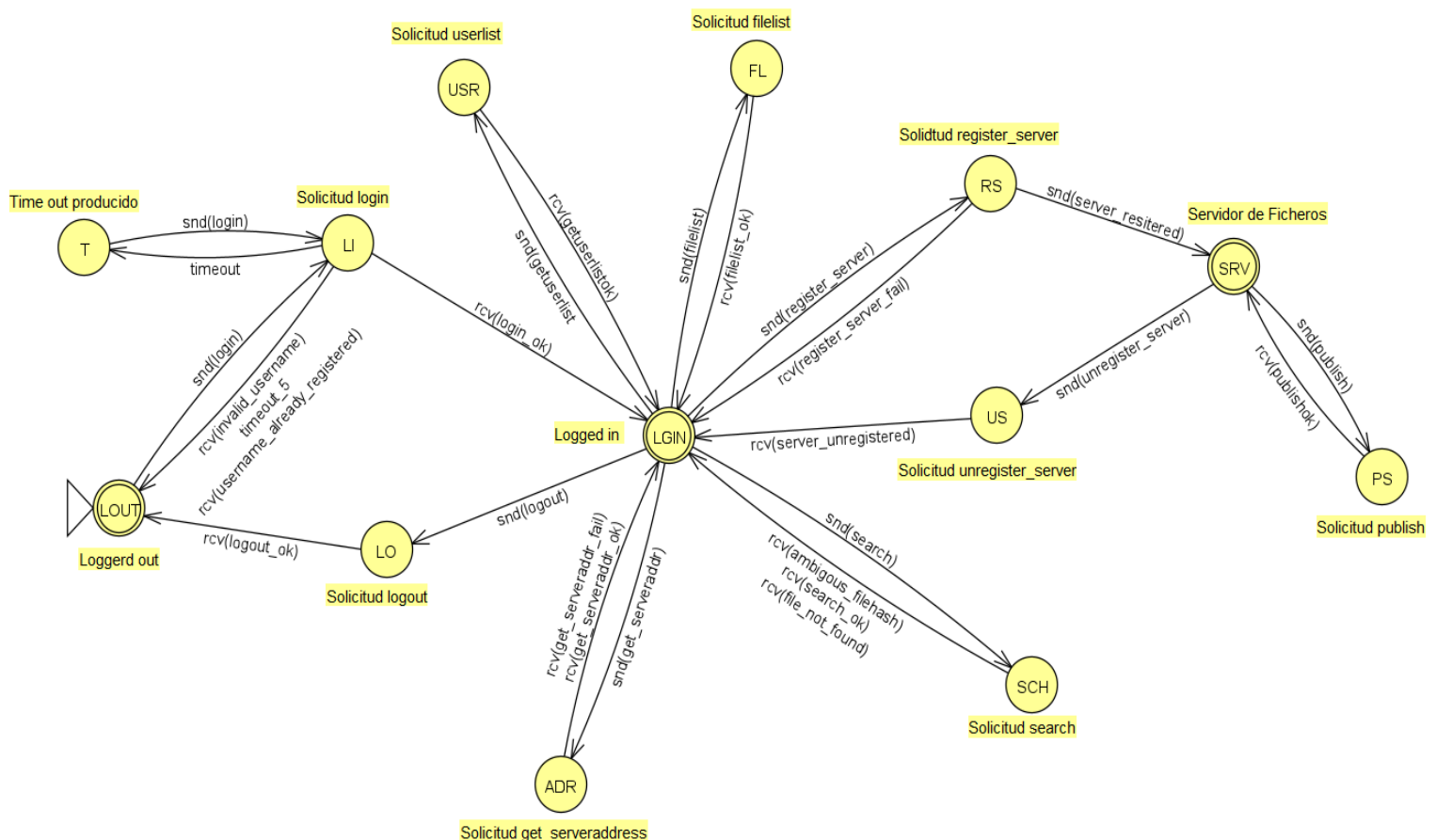
Opcode	Longitud del filehash	Filehash	Número de servidores que tienen el fichero	ID del servidor de ficheros
1 byte	1 byte	n bytes	1 byte	1 byte

### 3. Autómatas de protocolo

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

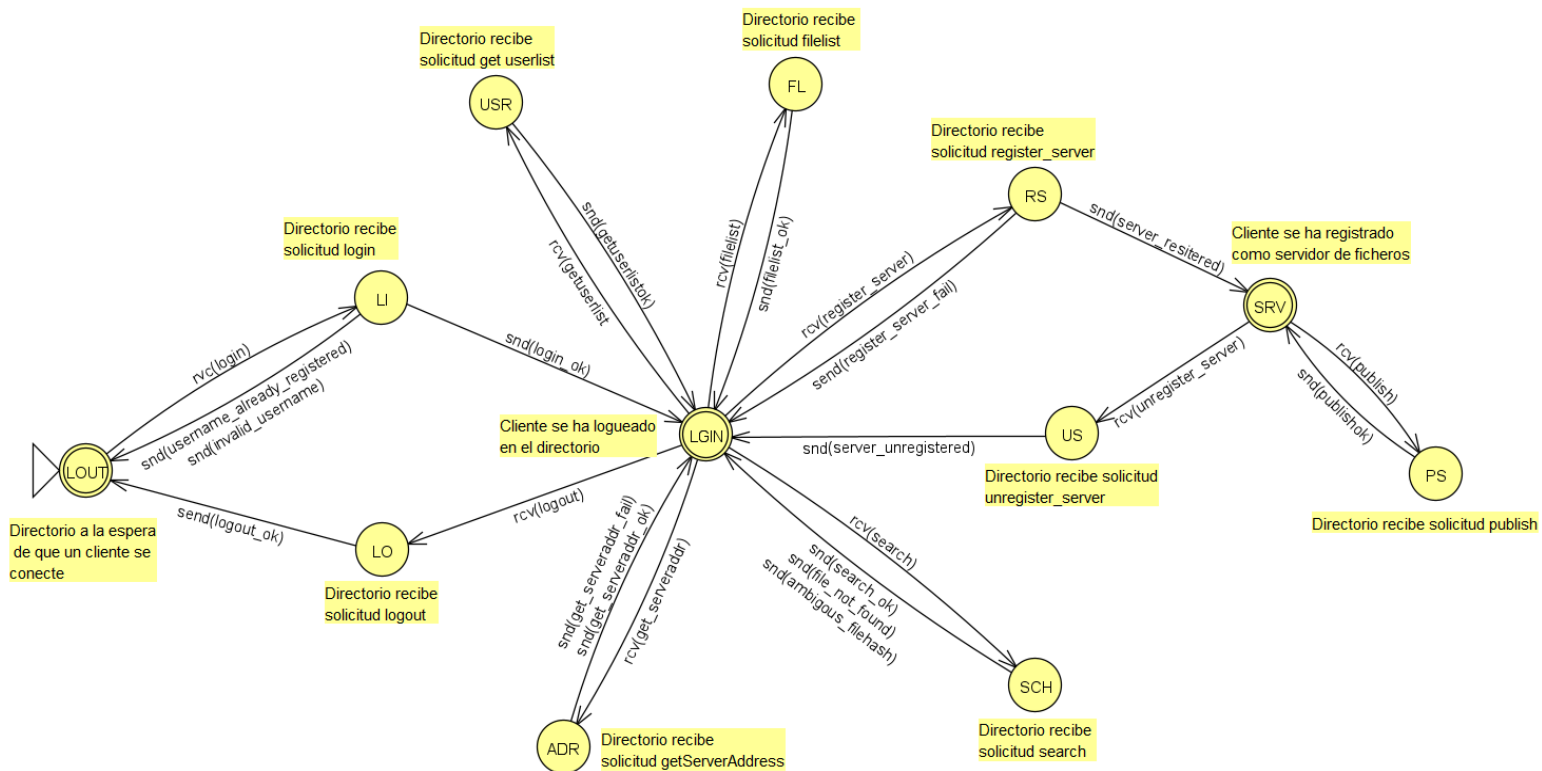
- Para cada operación, si el cliente no recibe una respuesta una vez pasado un tiempo determinado ocurre un **timeout**, reenviando el mensaje de solicitud de dicha operación. Esto lo hace hasta el quinto timeout, en el que da como fallado la operación. Sólo se ha mostrado cómo funciona mediante la operación login, pero funciona para cualquier otra.
- Nótese que el autómata no es completo, esto quiere decir que aquellas transiciones que no se han definido son imposibles de alcanzar en el código. Un ejemplo es realizar una solicitud login cuando ya se está logueado en el directorio.
- Aquellos **estados** que son **finales** representan estados del cliente que están a la espera de introducir órdenes en el shell (es decir, en estos estados **son los únicos desde los que se solicitan servicios**).
- Para no hacer más complejo el autómata, asumimos que **todas las operaciones que se hagan desde el estado LGIN (Logged In) también se pueden hacer desde el estado SRV (Server)**. No obstante, **las operaciones que se muestran en el autómata que se pueden hacer desde el estado SRV NO se pueden hacer en el estado LGIN** ni mucho menos en el estado LGOT (Logged Out).

Autómata rol cliente de directorio



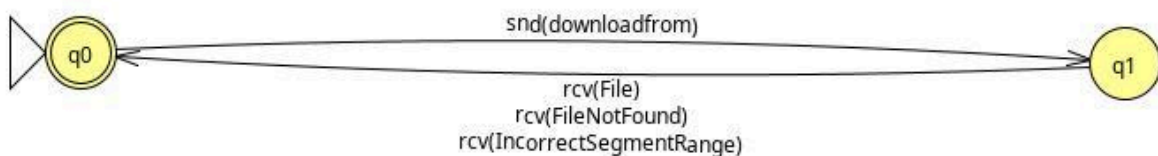
## Autómata rol servidor de directorio

- Este autómata representa la interacción entre un sólo cliente y el directorio. El directorio puede seguir escuchando solicitudes login mientras que sean de clientes distintos a los ya conectados en el directorio.
- Para este autómata seguimos considerando los dos últimos puntos mencionados anteriormente en el autómata del cliente.



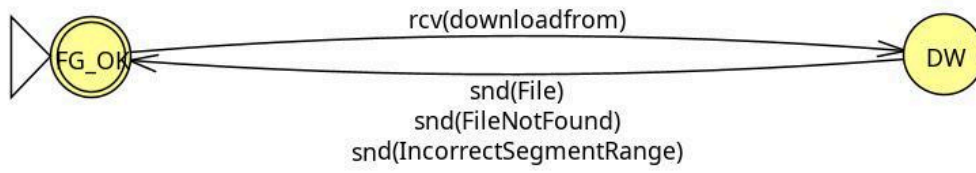
- Para los autómatas que representan el protocolo de comunicación entre dos personas, al sólo implementar la operación download from perteneciente a este protocolo, los autómatas quedan bastante simples, resumiendo en dos estados en ambos y prácticamente iguales.

## Autómata rol cliente de ficheros



---

### Autómata rol servidor de ficheros



## 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).

CLIENTE1: Solicita iniciar sesión

```
operation:login\n
nickname:alumno\n
\n
```

DIRECTORIO: Deniega el inicio de sesión

```
operation:loginfail\n
\n
```

CLIENTE1: Solicita inicio de sesión con un nickname distinto

```
operation:login\n
nickname:jiahui\n
\n
```

DIRECTORIO: Valida el inicio de sesión.

```
operation:loginok\n
sessionkey:3595\n
\n
```

CLIENTE2: Solicita inicio de sesión

```
operation:login\n
nickname:jiahui\n
\n
```

DIRECTORIO: Deniega el inicio de sesión, ya hay un cliente conectado con ese nickname.

```
operation:loginfail\n
\n
```

CLIENTE2: Solicita inicio de sesión

```
operation:login\n
nickname:jesus\n
\n
```

DIRECTORIO: Valida el inicio de sesión

```
operation:loginok\n
sessionkey:7964\n
\n
```

CLIENTE1: Solicita ver la lista de usuarios conectados

---

```
operation:getuserlist\n
sessionkey:3595\n
\n
```

DIRECTORIO: valida la lista de usuarios conectados

```
operation:getuserlistok\n
nickname:jesus,jiahui\n
\n
```

CLIENTE1: Solicita iniciar sesión

No puede ejecutar dicho comando porque ya está conectado. En su autómata no existe ninguna regla desde el estado Logged In (LI\_OK) que lo permita. Todo esto ocurre sin que el Directorio se entere, es decir, él sigue esperando a que le llegue algún mensaje.

CLIENTE1: Solicita cerrar sesión

```
operation:logout\n
sessionKey:3595\n
\n
```

DIRECTORIO: Valida la solicitud de cerrar sesión

```
operation:logoutok\n
\n
```

CLIENTE1: Solicita ver la lista de usuarios conectados.

No puede ejecutar dicho comando si no está conectado en el directorio. En el autómata, no existe una regla desde el estado Logged Out (LOGOUT) que permita el cambio al estado USR(estado que procesa la solicitud get userlist). No puede enviar entonces dicho mensaje y por lo tanto el Directorio sigue a la espera de la llegada de mensajes.

CLIENTE1: Solicita iniciar sesión

```
operation:login\n
nickname:jiahui\n
\n
```

DIRECTORIO: Valida el inicio de sesión

```
operation:loginok\n
sessionKey:1994\n
\n
```

CLIENTE1: Se convierte en bgserve, teniendo que registrarse en el directorio.

```
operation:register_server\n
sessionKey:1994\n
Port:algun puerto disponible entre 10000 y 65535\n
\n
```

DIRECTORIO: Recibe la solicitud y lo registra en el servidor, devolviendo una confirmación

```
operation:server_registered\n
\n
```



---

CLIENTE1: Publica los ficheros que tiene en su carpeta compartida

```
operation:publish\n
sessionKey:1994\n
filenames:nombre fichero1, nombre fichero2,... nombre fichero_i\n
filehashs:hash fichero1, hash fichero2,...hash fichero_i\n
filesizes:tamaño fichero1, tamaño fichero2,...,tamaño fichero_i\n
\n
```

DIRECTORIO: Envía el mensaje de confirmación de que se han publicado los ficheros

```
operation:publish_ok\n
\n
```

CLIENTE2: Solicita la lista de ficheros compartidos en el directorio

```
operation:filelist\n
sessionkey:7964\n
\n
```

DIRECTORIO: Envía un mensaje con la lista de ficheros compartidos en ese momento

```
operation:filelist_ok\n
filenames:nombre fichero1, nombre fichero2,... nombre fichero_i\n
filehashs:hash fichero1, hash fichero2,...hash fichero_i\n
filesizes:tamaño fichero1, tamaño fichero2,...,tamaño fichero_i\n
\n
```

CLIENTE2: Solicita la descarga de un fichero, pasando como argumento una parte del hash de uno de los ficheros compartidos por cliente 2. Primero se requiere conocer la dirección del cliente2.

```
operation:get_serveraddr\n
sessionkey:7964\n
nickname:jiahui\n
\n
```

DIRECTORIO: Recibe la solicitud y le devuelve la dirección del cliente 2. Tras ello, el cliente1 ya puede solicitarle la descarga directamente al servidor de ficheros cliente1.

```
operation:get_serveraddr_ok\n
server_adress: (IP:PORT del cliente1)\n
\n
```

Tras ello, cliente envía un mensaje de solicitud downloadfrom al servidor de ficheros con los siguientes datos:

```
opcode:1
hashLength:[numero entero]
Filehash:[según el ejemplo, una pequeña parte del hash de uno de los ficheros compartidos por cliente2]
```

---

Recibiendo por parte del servidor de ficheros el siguiente mensaje tipo File

```
opcode:2
Filehash: [hash completo del fichero descargado]
DownloadedFileLength: [tamaño del fichero descargado]
DownloadedFile: [Contenido del fichero]
```

CLIENTE2: Solicita conocer el número de servidores que tiene un fichero identificado por un hash

```
operation:search\n
sessionkey:7964\n
filehashs:[hash pasado por el cliente]\n
\n
```

DIRECTORIO: Directorio no es capaz de encontrar un fichero que haya sido compartido con el hash que ha pasado.

```
operation:file_not_found\n
\n
```

## 5. Implementación de las mejoras

---

- fgserve puerto variable: Para ello, lo que hemos hecho es tratar de crear una instancia NFServerSimple con un puerto, por defecto, el predeterminado(10000). Una vez se haya creado y lanzado el servidor de ficheros en primer plano, se genera un nuevo puerto entre 10000 y 65535.

En el caso de que el puerto esté ocupado se genera otro nuevo puerto. La probabilidad de generar dos veces un mismo puerto es  $(1/55535)^2$ , una probabilidad tan baja que se puede considerar practicamente imposible, por lo que no hemos implementado ningún mecanismo que se detenga dado cierto número de intentos.

- downloadfrom por nickname: Hemos implementado una nueva operación para ello, get Server Address, que dado un nombre de usuario, obtiene su dirección como servidor de ficheros. Para ello, hemos implementado en el Directorio una nueva estructura de datos que almacena para cada usuario que se lanza como servidor de ficheros su correspondiente dirección. Tras ello, el funcionamiento de downloadfrom es el mismo que el descrito en la implementación obligatoria.
- bgserve secuencial: Comprobamos antes de empezar que el servidor no se encuentra en marcha (Para ello, bgserve no se puede ejecutar una vez ya se ha ejecutado debido a que se pasa a un nuevo estado cuyo lanzamiento de dicha operación no es posible. Esto queda plasmado en el autómata). Una vez creado, se lanza, dejando el socket servidor a la espera de que un socket de un cliente se conecte para establecer conexión. Una vez un peer cliente establezca conexión, se lanza en el hilo principal el método de NFServerCom, serveFilesToClient(), encargado de tratar la solicitud de downloadfrom del peer cliente.
- bgserve multihilo: Tras la implementación del bgserve secuencial, el único cambio es que cada vez que se ha establecido conexión con un peer, se lanza una instancia NFServerThread, encargado únicamente a ejecutar el método serveFilesToClient(), es decir, a atender la solicitud de downloadfrom del cliente. Una vez hecho, dicho hilo cierra. Mientras tanto, el hilo principal sigue a la espera de conexiones.
- stopserver: Para dejar de ser un servidor de ficheros, se lanza un nuevo hilo, encargado únicamente a cerrar el socket servidor para así dejar de escuchar conexiones. En el caso de que esté una conexión en transcurso, el socket servidor no se cerrará hasta que termine.
- bgserve puerto efímero: Hemos seguido la misma idea del puerto variable de fgserve, con la diferencia de que antes de abrir a la escucha de conexiones el socket del servidor, se genera un puerto aleatorio. Con esto, si ocurre que dicho puerto seleccionado está siendo ocupado, se vuelve a generar un puerto aleatorio. De nuevo, no hemos incluido un mecanismo que se detenga una vez realizado un numero determinado de intentos (eso si, si no se puede abrir el socket por una razón ajena a que el puerto dado está en uso, se detiene), dado que con el segundo intento tenemos la misma probabilidad que la anterior definida.
- publish + filelist: Para la definición de publish, el servidor de ficheros que desea publicar sus ficheros de su carpeta compartida, requiere enviar un mensaje al directorio con el hash, el tamaño y el nombre de cada fichero. Una vez el directorio recibe el mensaje, crea las instancias FileInfo con los datos pasados por el cliente y los añade a una estructura de datos del directorio que almacena para cada usuario, los ficheros que ha subido.

Cuando un usuario solicita la lista de ficheros, una vez enviado la solicitud al directorio y este lo recibe, el directorio accede a la estructura anteriormente mencionada e imprime todos ficheros no repetidos (para ello, hemos redefinido los métodos equals() y hashCode() de la clase FileInfo, de manera que dos **ficheros son iguales si comparten el mismo nombre, contenido y tamaño**)

- 
- publish + search: **Consideramos que dos servidores comparten el mismo fichero si este fichero contiene el mismo hash y mismo tamaño, sin importar su nombre.** Para ello, una vez recibido el directorio la solicitud, obtiene aquellos ficheros que tienen el mismo hash (o en caso de que sea un subhash, que lo contengan en su hash completo). Tras ello, se comprueba que de estos ficheros obtenidos, todos tengan el mismo hash, en el caso de que no sea así, el filehash es ambiguo, y si directamente no se ha obtenido un fichero es que el hash no identifica a ningún fichero. Una vez se ha comprobado que todos los ficheros obtenidos tienen el mismo hash (o solo se ha obtenido un único fichero), se usa la estructura de datos que almacena para cada usuario los ficheros que ha publicado, comprobando para cada usuario si han publicado alguno de los ficheros obtenidos anteriormente. Tras ello, se crea un mensaje de confirmación con los usuarios que hayan publicado ese fichero.
  - Baja ficheros y servidores: Cada vez que un cliente conectado deja de ser servidor de ficheros, se actualizan todas sus estructuras de datos, eliminando de la estructura de datos que almacena para cada usuario sus ficheros publicados la entrada correspondiente a este cliente.
  - userlist con servidores: El directorio, ahora al enviar la lista de usuarios que hay conectados en ese momento, revisa para cada usuario si es un servidor de ficheros y si lo es, añade al final del nickname un asterisco. A la hora de imprimir el mensaje, si el usuario ve que un nickname contiene un asterisco, entenderá que es un servidor registrado en el directorio, en caso contrario entiende que no lo es.
  - filelist ampliado con servidores: Se ha añadido un nuevo atributo a la clase FileInfo que guarda la lista de servidores registrados en ese momento por el usuario. El directorio ahora contiene una nueva estructura de datos que por cada filehash registrado, se tiene una lista con todos los usuarios que lo han publicado. Ahora, cada vez que se envíe un tipo de mensaje filelist ok, contiene en el campo nickname, una lista de usuarios por cada fichero que ha sido publicado y separado cada lista entre comas. Para separar los usuarios que han publicado un mismo fichero, se ha usado como separador de nuevo el asterisco.
  - download secuencial: Sigue de forma parecida el funcionamiento del downloadfrom con la diferencia de que sólo descargará un fragmento del fichero. Este fragmento estará definido dependiendo del número de servidores que tengan el fichero(n) y del identificador que tenga dicho servidor (1,2,3...n). Al realizarse de forma secuencial, la descarga se hace de forma ordenada, es decir, se van descargando los fragmentos de forma ordenada para una vez estén descargados se puedan escribir sin necesidad de guardarlos previamente en un buffer a la espera de que lleguen sus anteriores.
  - Comando fgstop: Como una vez se encuentre en modo de servidor en primer plano deja de leer del shell, hacemos que lea del buffer de entrada. Esto se realiza cada vez que transcurra el tiempo estipulado de timeout para recibir un paquete. Si la entrada en el buffer en ese momento coincide con "fgstop", deja de servir en primer plano.

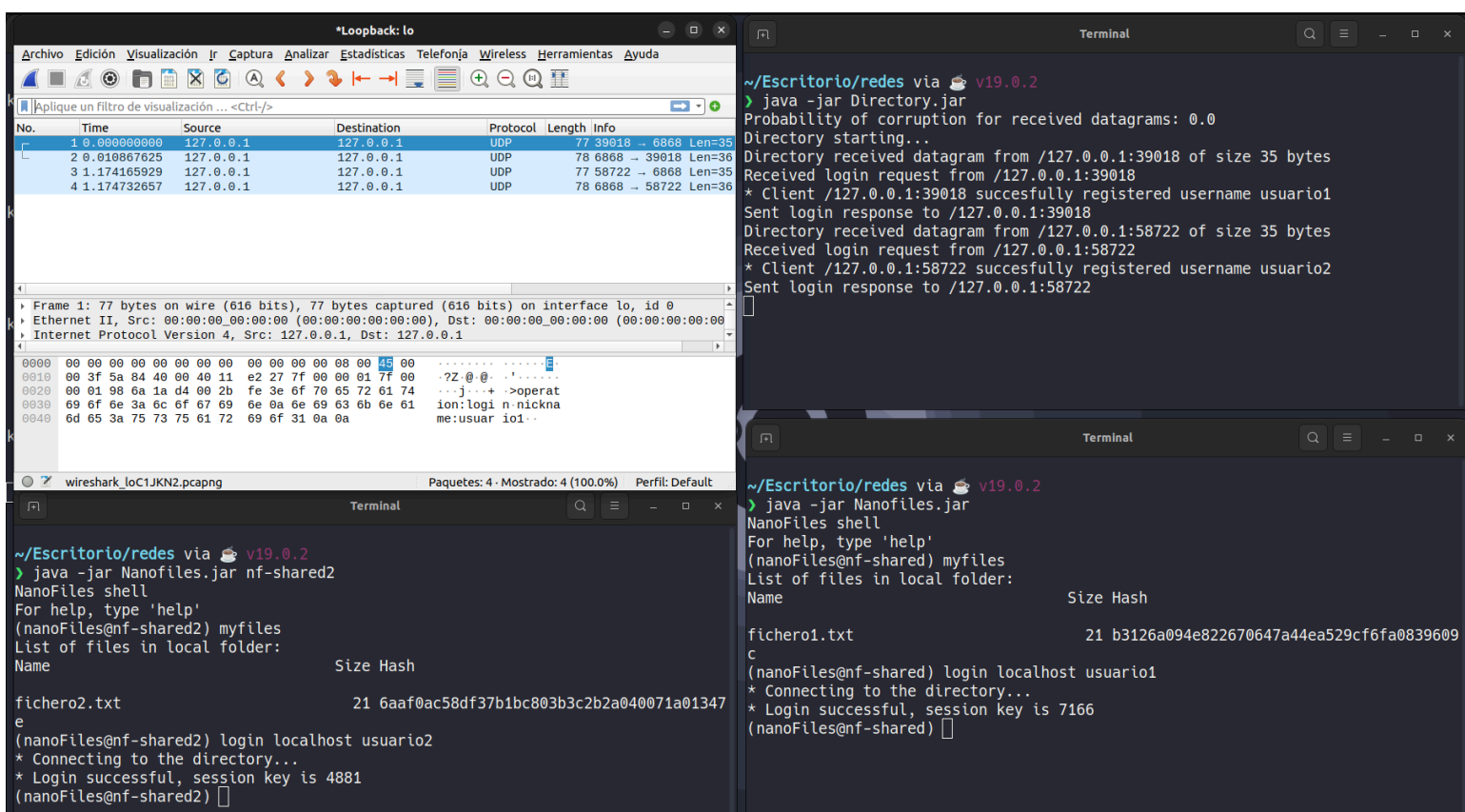
## 6. Ejemplo Wireshark

### 1.- Comando ejecutado: login

La primera instrucción que ejecutamos después de arrancar el servidor y los dos clientes es ejecutar el comando `login` para iniciar sesión, y darse de alta en el servidor.

Se puede observar la comunicación con el directorio mediante el protocolo UDP en los paquetes capturados por wireshark. En este caso un mensaje de cliente a servidor pidiendo iniciar sesión y otro de respuesta de parte del servidor confirmando la misma y devolviendo la session key, al ejecutar dos `login`, tenemos 2 pares de estos paquetes.

Además, se puede observar otro comando, el `myfiles`, que se ejecuta para leer el directorio local del cliente y mostrar los ficheros del mismo.



The screenshot displays a Wireshark network traffic analysis window and two terminal windows. The Wireshark window shows a capture on the loopback interface `lo` with four UDP packets. The first two packets are from `127.0.0.1` to `127.0.0.1` on port `39018`, and the next two are from `127.0.0.1` to `127.0.0.1` on port `58722`. The packet details show the Ethernet II, Internet Protocol Version 4, and Internet Protocol Version 4 fields. The packet bytes show the raw data, including the session key and the username.

The top terminal window shows the output of the `Directory.jar` application. It displays the directory starting, receiving login requests from `/127.0.0.1:39018` and `/127.0.0.1:58722`, and sending login responses to both clients. The output also shows the session key and the username for each client.

The bottom terminal window shows the output of the `NanoFiles.jar` application. It displays the NanoFiles shell, the help text, and the output of the `myfiles` command. The output shows the list of files in the local folder, including `fichero1.txt` and `fichero2.txt`, with their sizes and hashes.

## 2.- Comando ejecutado: bgserve en el usuario1

Este comando se ejecuta para convertir el host en un servidor de ficheros en segundo plano. De este modo, habilitamos la publicación de los ficheros locales del usuario para ponerlos a disposición de otros usuarios que estén conectados. Para ello, al realizar bgserve, se envía un paquete UDP al directorio en el que se solicita su registro como servidor para que los clientes puedan descargar ficheros usando el nickname del servidor.

La comunicación es cliente - servidor de modo que se emplea el protocolo UDP como muestra wireshark.

PD:(Los paquetes DNS capturados pertenecen a otro programa, no Nanofiles)

The screenshot displays a network capture in Wireshark and two terminal windows. The Wireshark window shows two UDP packets between 127.0.0.1 and 127.0.0.1. The first terminal window shows the output of `java -jar Directory.jar`, which acts as a directory server. The second terminal window shows the output of `java -jar NanoFiles.jar nf-shared2`, which acts as a file server.

**Wireshark Packet List:**

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	164	39018 → 6868 Len=122
2	0.001357618	127.0.0.1	127.0.0.1	UDP	64	6868 → 39018 Len=22

**Terminal 1 (Directory.jar):**

```
~/Escritorio/redes via v19.0.2
> java -jar Directory.jar
Probability of corruption for received datagrams: 0.0
Directory starting...
Directory received datagram from /127.0.0.1:39018 of size 35 bytes
Received login request from /127.0.0.1:39018
* Client /127.0.0.1:39018 successfully registered username usuario1
Sent login response to /127.0.0.1:39018
Directory received datagram from /127.0.0.1:58722 of size 35 bytes
Received login request from /127.0.0.1:58722
* Client /127.0.0.1:58722 successfully registered username usuario2
Sent login response to /127.0.0.1:58722
Directory received datagram from /127.0.0.1:39018 of size 54 bytes
Received register_server request from /127.0.0.1:39018
* Client /127.0.0.1:39018 (usuario1) successfully registered as server
Directory received datagram from /127.0.0.1:39018 of size 122 bytes
Received publish_files request from /127.0.0.1:39018
Sent publish_files request from /127.0.0.1:39018
```

**Terminal 2 (NanoFiles.jar):**

```
~/Escritorio/redes via v19.0.2
> java -jar NanoFiles.jar nf-shared2
NanoFiles shell
For help, type 'help'
(nanoFiles@nf-shared2) myfiles
List of files in local folder:
Name                                     Size Hash
fichero1.txt                             21 b3126a094e822670647a44ea529cf6fa0839609
(nanoFiles@nf-shared2) login localhost usuario1
* Connecting to the directory...
* Login successful, session key is 7166
(nanoFiles@nf-shared2) bgserve
* You are now serving files on port 37128
* NFServer server running on 0.0.0.0/0.0.0.0:37128
* File server registered with directory
(nanoFiles@nf-shared2) publish
* List of local files published to the directory
(nanoFiles@nf-shared2)
```

### 3.- Comando ejecutado: publish en el usuario1

Este mensaje se envía mediante el protocolo UDP como los anteriores para notificar al servidor de que hemos arrancado el servidor de ficheros en segundo plano y que ponemos a disposición de los demás, los ficheros de nuestro directorio local. De esta forma, el directorio registra el fichero, y puede mostrárselo a los demás usuarios en caso de que lo pidan.

The image displays a network capture in Wireshark and two terminal windows. The Wireshark capture shows traffic on the loopback interface 'lo' between 127.0.0.1 and 127.0.0.53. The selected packet is a DNS Standard query response (No. 6, Time 0.861282754) from 127.0.0.53 to 127.0.0.1. The packet details show it's a response to a query for 'ion:serv er\_regis'.

The top terminal window shows the execution of the 'publish' command in the 'usuario1' shell. The output indicates that the directory received a datagram from /127.0.0.1:39018, successfully registered the username 'usuario1', and sent a login response to /127.0.0.1:39018.

The bottom terminal window shows the execution of the 'publish' command in the 'usuario2' shell. The output indicates that the directory received a datagram from /127.0.0.1:58722, successfully registered the username 'usuario2', and sent a login response to /127.0.0.1:58722.

The bottom terminal window also shows the execution of the 'publish' command in the 'usuario1' shell. The output indicates that the directory received a datagram from /127.0.0.1:39018, successfully registered the username 'usuario1', and sent a login response to /127.0.0.1:39018.

## 4.- Comando ejecutado: filelist en el usuario2

Este comando que se transmite mediante UDP, sirve para pedirle al directorio que muestre los ficheros disponibles publicados por otros usuarios. En este ejemplo el directorio devuelve *fichero1.txt* publicado anteriormente por el usuario1.

The image displays a network traffic capture in Wireshark and the corresponding terminal output for a file sharing application.

**Wireshark Packet List:**

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	78	58722 → 6868 Len=36
2	0.001472418	127.0.0.1	127.0.0.1	UDP	152	6868 → 58722 Len=116

**Terminal Output (Left):**

```
NanoFiles shell
For help, type 'help'
(nanoFiles@nf-shared2) myfiles
List of files in local folder:
Name                               Size Hash
fichero2.txt                        21 6aaf0ac58df37b1bc803b3c2b2a040071a01347
e
(nanoFiles@nf-shared2) login localhost usuario2
* Connecting to the directory...
* Login successful, session key is 4881
(nanoFiles@nf-shared2) filelist
* These are the files tracked by the directory localhost/127.0.0.1:6868
Name                               Size Hash
fichero1.txt                        21 b3126a094e822670647a44ea529cf6fa0839609
c
(nanoFiles@nf-shared2)
```

**Terminal Output (Right):**

```
Directory starting...
Directory received datagram from /127.0.0.1:39018 of size 35 bytes
Received login request from /127.0.0.1:39018
* Client /127.0.0.1:39018 succesfully registered username usuario1
Sent login response to /127.0.0.1:39018
Directory received datagram from /127.0.0.1:58722 of size 35 bytes
Received login request from /127.0.0.1:58722
* Client /127.0.0.1:58722 succesfully registered username usuario2
Sent login response to /127.0.0.1:58722
Directory received datagram from /127.0.0.1:39018 of size 54 bytes
Received register_server request from /127.0.0.1:39018
* Client /127.0.0.1:39018 (usuario1) successfully registered as server
Directory received datagram from /127.0.0.1:39018 of size 122 bytes
Received publish_files request from /127.0.0.1:39018
Sent publish_files request from /127.0.0.1:39018
Directory received datagram from /127.0.0.1:58722 of size 36 bytes
Received filelist request from /127.0.0.1:58722
Sent filelist response to /127.0.0.1:58722

~/Escritorio/redes via v19.0.2
> java -jar Nanofiles.jar
NanoFiles shell
For help, type 'help'
(nanoFiles@nf-shared) myfiles
List of files in local folder:
Name                               Size Hash
fichero1.txt                        21 b3126a094e822670647a44ea529cf6fa0839609
c
(nanoFiles@nf-shared) login localhost usuario1
* Connecting to the directory...
* Login successful, session key is 7166
(nanoFiles@nf-shared) bgserve
* You are now serving files on port 37128
* NFServe server running on 0.0.0.0/0.0.0.0:37128
* File server registered with directory
(nanoFiles@nf-shared) publish
* List of local files published to the directory
(nanoFiles@nf-shared)
```



## 5.- Comando ejecutado: downloadfrom en el usuario1

Este comando lo ejecuta el usuario 1 para descargar el fichero que está disponible en el directorio. En este caso descarga el fichero *fichero1.txt* ofrecido por el usuario1. Los ficheros se identifican por el hash code.

Como se puede apreciar, en este caso al no proporcionarle la ip del servidor de ficheros, primero se envía el mensaje al directorio que se encarga de encontrar la ip del servidor y se lo notifica al cliente que lo solicitó. Ya con este dato se establece la conexión con el servidor de ficheros mediante TCP. Podemos observar cómo se lleva a cabo una sincronización con los dos primeros paquetes SYN y posteriormente los paquetes TCP con los datagramas y el ACK de confirmación.

The image displays two windows: Wireshark on the left and a Terminal on the right.

**Wireshark Window:** Shows a packet capture on interface 'lo'. The packet list table is as follows:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	86	58722 → 6868 Len=44
2	0.001232209	127.0.0.1	127.0.0.1	UDP	103	6868 → 58722 Len=61
3	0.019143934	127.0.0.1	127.0.0.1	TCP	74	56940 → 37128 [SYN]
4	0.019184626	127.0.0.1	127.0.0.1	TCP	74	37128 → 56940 [SYN]
5	0.019215988	127.0.0.1	127.0.0.1	TCP	66	56940 → 37128 [ACK]
6	0.026980417	127.0.0.1	127.0.0.1	TCP	66	56940 → 37128 [PSH]
7	0.026991027	127.0.0.1	127.0.0.1	TCP	66	37128 → 56940 [ACK]
8	0.027834624	127.0.0.1	127.0.0.1	TCP	66	56940 → 37128 [PSH]
9	0.027837744	127.0.0.1	127.0.0.1	TCP	66	37128 → 56940 [ACK]
10	0.027869551	127.0.0.1	127.0.0.1	TCP	106	56940 → 37128 [PSH]
11	0.027876455	127.0.0.1	127.0.0.1	TCP	66	37128 → 56940 [ACK]
12	0.027541643	127.0.0.1	127.0.0.1	TCP	67	37128 → 56940 [PSH]
13	0.027546662	127.0.0.1	127.0.0.1	TCP	66	56940 → 37128 [ACK]
14	0.027574824	127.0.0.1	127.0.0.1	TCP	67	37128 → 56940 [PSH]
15	0.027576495	127.0.0.1	127.0.0.1	TCP	66	56940 → 37128 [ACK]
16	0.027602295	127.0.0.1	127.0.0.1	TCP	106	37128 → 56940 [PSH]
17	0.027604293	127.0.0.1	127.0.0.1	TCP	66	56940 → 37128 [ACK]
18	0.027618181	127.0.0.1	127.0.0.1	TCP	67	37128 → 56940 [PSH]
19	0.027620021	127.0.0.1	127.0.0.1	TCP	66	56940 → 37128 [ACK]
20	0.027634054	127.0.0.1	127.0.0.1	TCP	87	37128 → 56940 [PSH]
21	0.027635917	127.0.0.1	127.0.0.1	TCP	66	56940 → 37128 [ACK]

The packet details pane for packet 21 shows:

- Frame 21: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface lo, id 0
- Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- Transmission Control Protocol, Src Port: 56940, Dst Port: 37128, Seq: 43, Ack: 65, Len: 0

The packet bytes pane shows the raw data: 0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00 .....E:

**Terminal Window:** Shows the execution of the NanoFiles application.

```
~/Escritorio/redes via v19.0.2
> java -jar Nanofiles.jar
NanoFiles shell
For help, type 'help'
(nanoFiles@nf-shared) myfiles
List of files in local folder:
Name                                     Size Hash
fichero1.txt                             21 b3126a094e822670647a44ea529cf6fa0839609c
(nanoFiles@nf-shared) login localhost usuario1
* Connecting to the directory...
* Login successful, session key is 7166
(nanoFiles@nf-shared) bgserve
* You are now serving files on port 37128
* NFServe server running on 0.0.0.0/0.0.0.0:37128
* File server registered with directory
(nanoFiles@nf-shared) publish
* List of local files published to the directory
(nanoFiles@nf-shared) New Client connected from /127.0.0.1:37128
Disconnected client from /127.0.0.1:37128
```

The second terminal window shows the execution of the `downloadfrom` command:

```
(nanoFiles@nf-shared2) downloadfrom usuario1 b3126a094e822670647a44ea529cf6fa0839609c
Correct use:downloadfrom <nickname/IP:port> <file_hash> <local_filename>
(nanoFiles@nf-shared2) downloadfrom usuario1 b3126a094e822670647a44ea529cf6fa0839609c fichero1.txt
Connected to /127.0.0.1:37128
Starting download
Successfully downloaded remote file to /home/jiahui/Escritorio/redes/nf-shared2/fichero1.txt
File 'fichero1.txt' downloaded succesfully.
(nanoFiles@nf-shared2)
```

## 7. Conclusiones

---

### **Jiahui Lin:**

En mi opinión, ha sido un proyecto muy complejo y que definitivamente nos ha supuesto un reto realizar. Por mi parte diría que ha servido de ayuda para comprender algunos conceptos de teoría y entender mejor el paso de mensajes entre hosts y servidores y p2p. Además, la implementación de los distintos protocolos de nivel de transporte como UDP y TCP me ha ayudado a afianzar mi conocimiento sobre los mismos de la teoría a la práctica. En definitiva, un proyecto interesante pero francamente complejo.

### **Jesús Sánchez Pardo:**

Desde mi punto de vista, este proyecto ha sido un gran desafío y una gran manera de comprender el funcionamiento de los protocolos UDP y TCP, como se establecen conexiones entre dos hosts/dos peers, principalmente a la implementación de las mejoras como search y bgserve, las mejoras que más nos ha dado problemas debido al formato de los mensajes enviados entre cliente-directorio. Opino, y seguramente mi compañero opinará lo mismo, que este proyecto ha sido difícil, pero nos ha aportado un mejor entendimiento de la asignatura.