MEMORIA PROYECTO

NANOFILES

REDES DE COMUNICACIONES

Grupo 1.1 – Curso 2023/2024

Realizado por:

Alberto Crespo Palma: 49177961M

Daniel Conesa Sáez: 24450042F

ÍNDICE

Introducción

En este documento se explica el diseño de la aplicación *NanoFiles*, una aplicación dedicada a la transferencia de archivos. Se detallará el diseño e implementación de los mensajes utilizados para los protocolos (además de capturarlos a través de Wireshark) acompañados de los autómatas utilizados para el correcto funcionamiento de la aplicación. También se hablará de las mejoras incluidas en el proyecto y, por último, las conclusiones obtenidas tras haberlo terminado.

Protocolos diseñados

Esta aplicación utiliza los protocolos TCP y UDP para el intercambio de mensajes. El protocolo UDP es usado generalmente para comunicarse con el directorio, mientras que el protocolo TCP se usa para la transferencia de archivos entre hosts (P2P).

Directorio

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

Tipos y descripción de los mensajes:

Comando login

Mensaje: login

Sentido de la comunicación: Cliente→ Directorio

Descripción: mensaje enviado por el cliente de NanoFiles para iniciar sesión con el nombre indicado como parámetro.

Ejemplo:

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

Mensaje: login_ok

Sentido de la comunicación: Directorio→ Cliente

Descripción: mensaje enviado por el directorio para indicar que el inicio de sesión se realizó con éxito. Devuelve la Session Key al cliente.

Ejemplo:

```
operation:loginok\n
nickname:alumno\n
sessionkey:63452\n
\n
```

Mensaje: login_failed

Sentido de la comunicación: Directorio→ Cliente

Descripción: mensaje enviado por el directorio para indicar que el inicio de sesión no se realizó.

Ejemplo:

```
operation:loginfailed\n
nickname:alumno\n
\n
```

Comando logout

Mensaje: logout

Sentido de la comunicación: Cliente→ Directorio

Descripción: mensaje enviado por el cliente para desconectarse del servidor.

Ejemplo:

```
operation:logout\n
sessionkey:63452\n
\n
```

Mensaje: logout_ok

Sentido de la comunicación: Directorio→ Cliente

Descripción: mensaje enviado por el directorio para indicar que que el logout se realizó con éxito.

Ejemplo:

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

Comando userlist

Mensaje: userlist

Sentido de la comunicación: Cliente→ Directorio

Descripción: mensaje enviado por el cliente para pedir una lista de los usuarios con sesión iniciada en el momento.

Ejemplo:

```
operation:userlist\n
\n
```

Mensaje: userlist_ok

Sentido de la comunicación: Directorio→ Cliente

Descripción: mensaje enviado por el directorio para devolver la lista de usuarios.

Ejemplo:

```
operation:userlistok\n
userlist:alumno,admin,alberto,dani\n
\n
```

Comando bgserve

Mensaje: register_server_port

Sentido de la comunicación: Cliente→ Directorio

Descripción: mensaje enviado por el cliente para registrar el puerto de su servidor en el directorio tras haber iniciado un servidor. Este puerto se asocia a una session key que está logueado en el directorio.

Ejemplo:

```
operation:register_server_port\n
sessionkey:63452\n
port:14320\n
\n
```

Mensaje: server_port_registered

Sentido de la comunicación: Directorio→ Cliente

Descripción: mensaje enviado por el directorio para indicar que el puerto se registró con éxito y se asoció al cliente que lo envió.

Ejemplo:

```
operation:server_port_registered\n
\n
```

Mensaje: server_port_failed

Sentido de la comunicación: Directorio→ Cliente

Descripción: mensaje enviado por el directorio para indicar que el servidor de ficheros no se registró correctamente.

Ejemplo:

```
operation:server_port_failed\n
\n
```

Comando stopserver

Mensaje: unregister_server

Sentido de la comunicación: Cliente→ Directorio

Descripción: mensaje enviado por el cliente para eliminar el puerto asociado a su session key del directorio.

Ejemplo:

```
operation:unregister_server\n
sessionkey:63452\n
\n
```

Mensaje: server_unregistered

Sentido de la comunicación: Directorio→ Cliente

Descripción: mensaje enviado por el directorio para indicar que el puerto fue eliminado y desasociado de la session key del cliente correspondiente.

Ejemplo:

```
operation:server_unregistered\n
\n
```

Comando downladfrom “nickname”

Mensaje: lookup_address

Sentido de la comunicación: Cliente→ Directorio

Descripción: mensaje enviado por el cliente para preguntar si hay alguna IP asociada al nickname enviado en el directorio.

Ejemplo:

```
operation:lookup_address\n
nickname:alumno\n
\n
```

Mensaje: send_address

Sentido de la comunicación: Directorio→ Cliente

Descripción: mensaje enviado por el directorio para indicar que encontró una IP asociada al nickname que se envió.

Ejemplo:

```
operation:send_address\n
address:/192.168.1.54:10564\n
\n
```

Mensaje: address_not_found

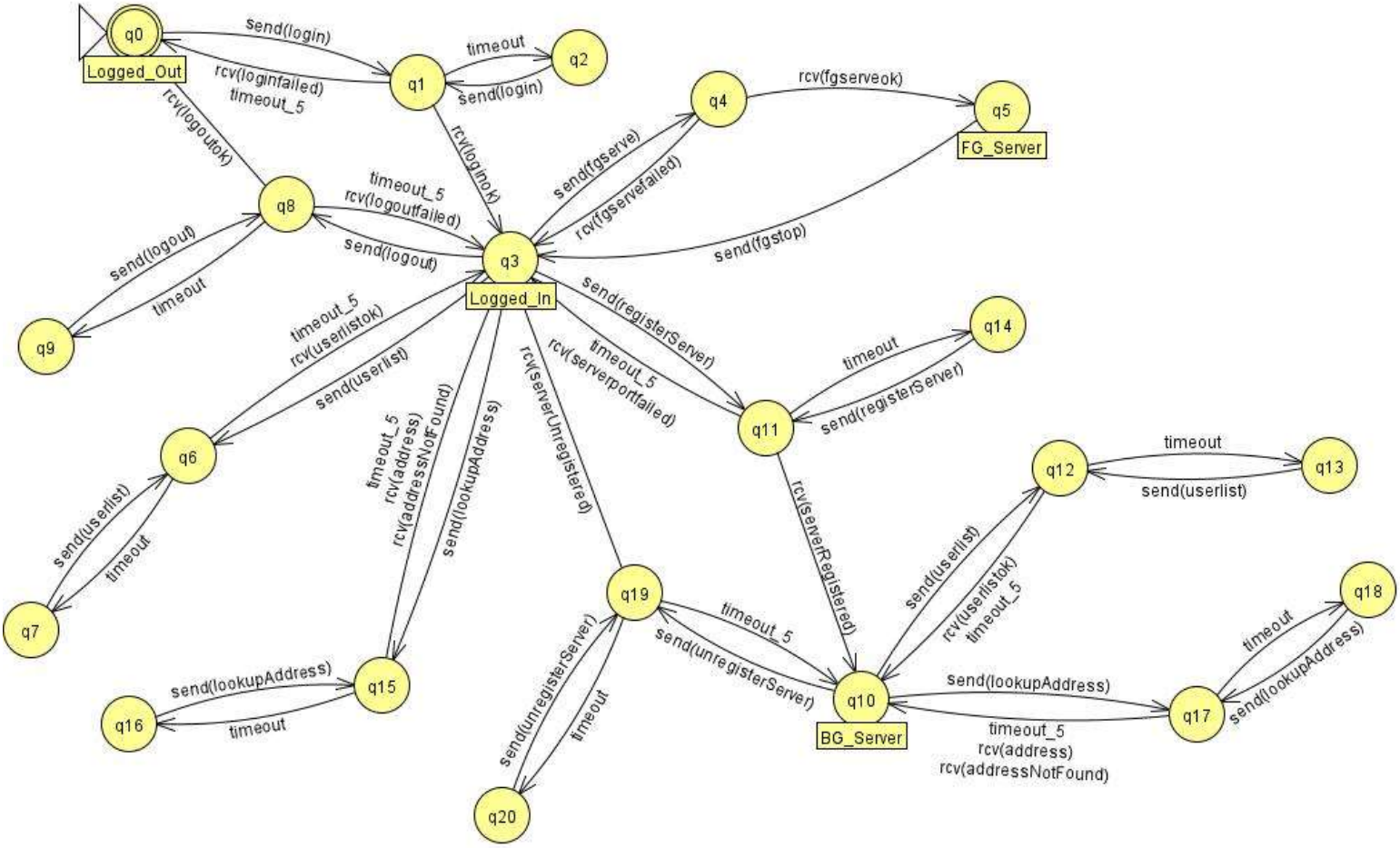
Sentido de la comunicación: Directorio→ Cliente

Descripción: mensaje enviado por el directorio para indicar que no encontró ninguna IP asociada al nickname que se envió.

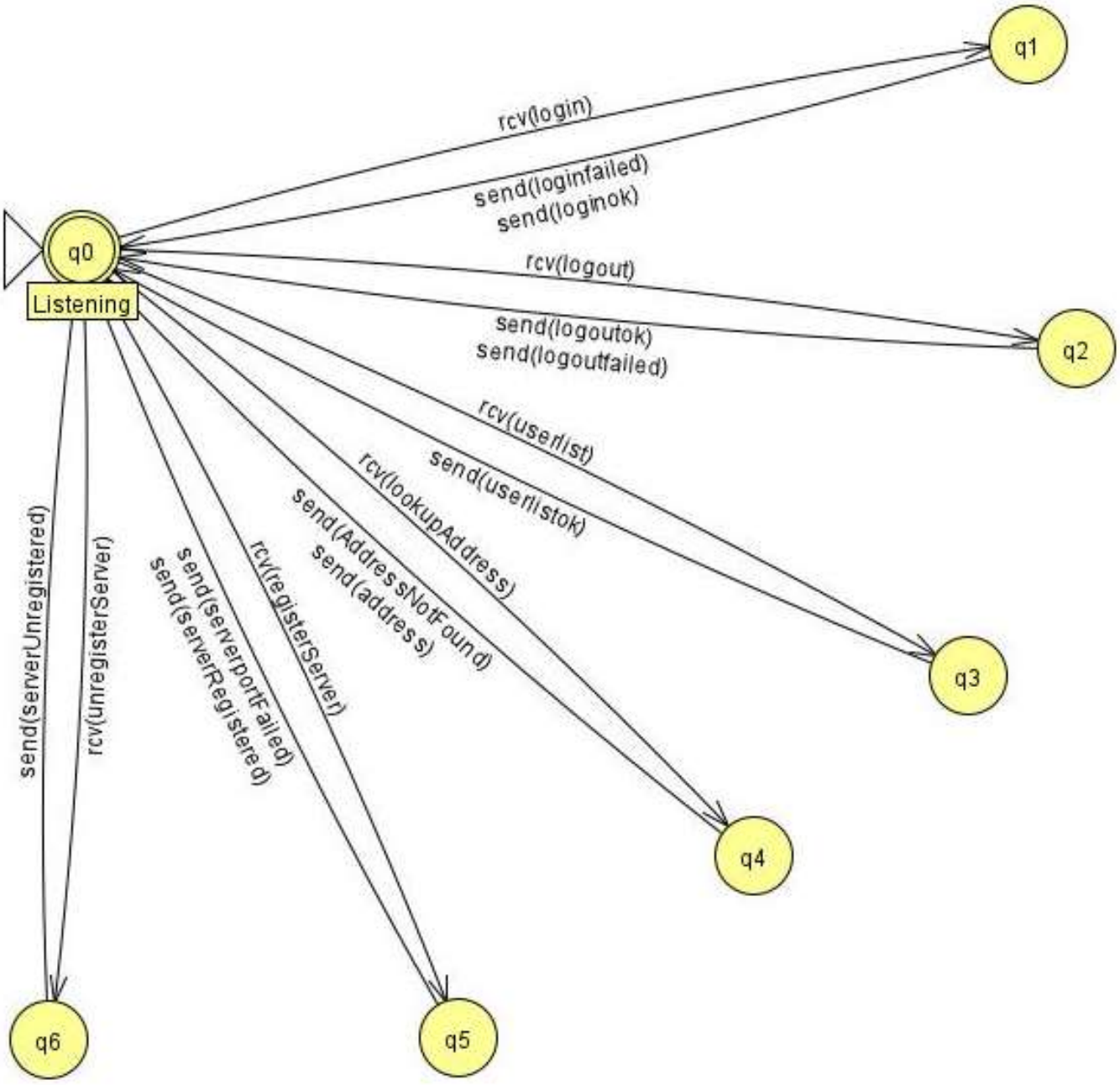
Ejemplo:

```
operation:address_not_found\n
\n
```

Autómata rol cliente de directorio:



Autómata rol servidor de directorio:



Peer-to-peer

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

Formato: Control

Descripción: Este mensaje lo envía el par servidor de ficheros al par cliente (receptor) de fichero para indicar que el mensaje que es recibido no se puede tratar.

Ejemplo:

Opcode (1 byte)
0

Mensaje: FileNotFound (opcode = 1)

Sentido de la comunicación: Servidor de ficheros → Cliente

Formato: Control

Descripción: Este mensaje lo envía el par servidor de ficheros al par cliente (receptor) de ficheros 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: TransferFile (opcode = 2)

Sentido de la comunicación: Servidor de ficheros → Cliente

Formato: TLV

Descripción: Este mensaje lo envía el servidor de ficheros al cliente de ficheros para enviar un archivo (usado para comando *downloadfrom*).

Ejemplo:

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

Mensaje: DownloadFromRequest (opcode = 3)

Sentido de la comunicación: Cliente → Servidor de ficheros

Formato: TLV

Descripción: Este mensaje lo envía el par cliente al par servidor de ficheros para solicitar descargar un archivo (usado para comando downloadfrom).

Ejemplo:

Opcode (1 byte)	Longitud (8 bytes)	Hash del fichero o subcadena (n bytes)
3	**n**	**x**

Mensaje: SendFileHash (opcode = 4)

Sentido de la comunicación: Servidor de ficheros → Cliente

Formato: TLV

Descripción: Este mensaje es enviado por el servidor de ficheros para, posteriormente, poder hacer una comparación de hashes del fichero descargado con el fichero original (usado para comando *downloadfrom*).

Ejemplo:

Opcode (1 byte)	Longitud (8 bytes)	Hash del fichero o subcadena (n bytes)
4	**n**	**x**

Mensaje: HashReceived (opcode = 5)

Sentido de la comunicación: Cliente → Servidor de ficheros

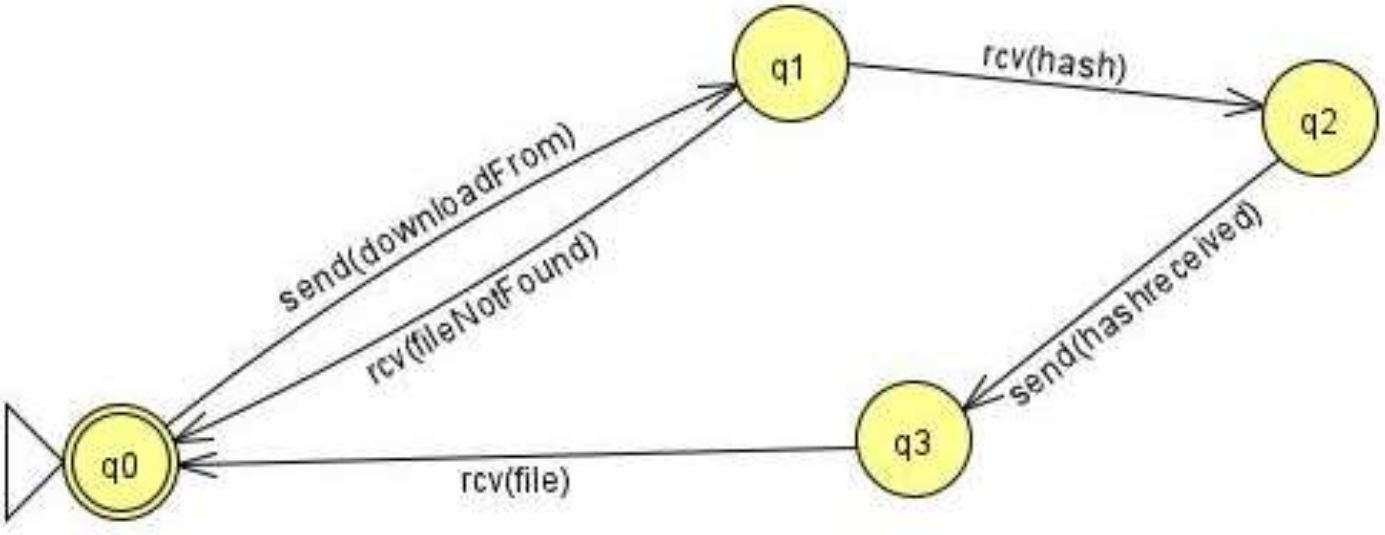
Formato: Control

Descripción: Este mensaje es enviado por el cliente para indicarle al servidor de ficheros que le ha llegado el hash (usado para comando *downloadfrom*).

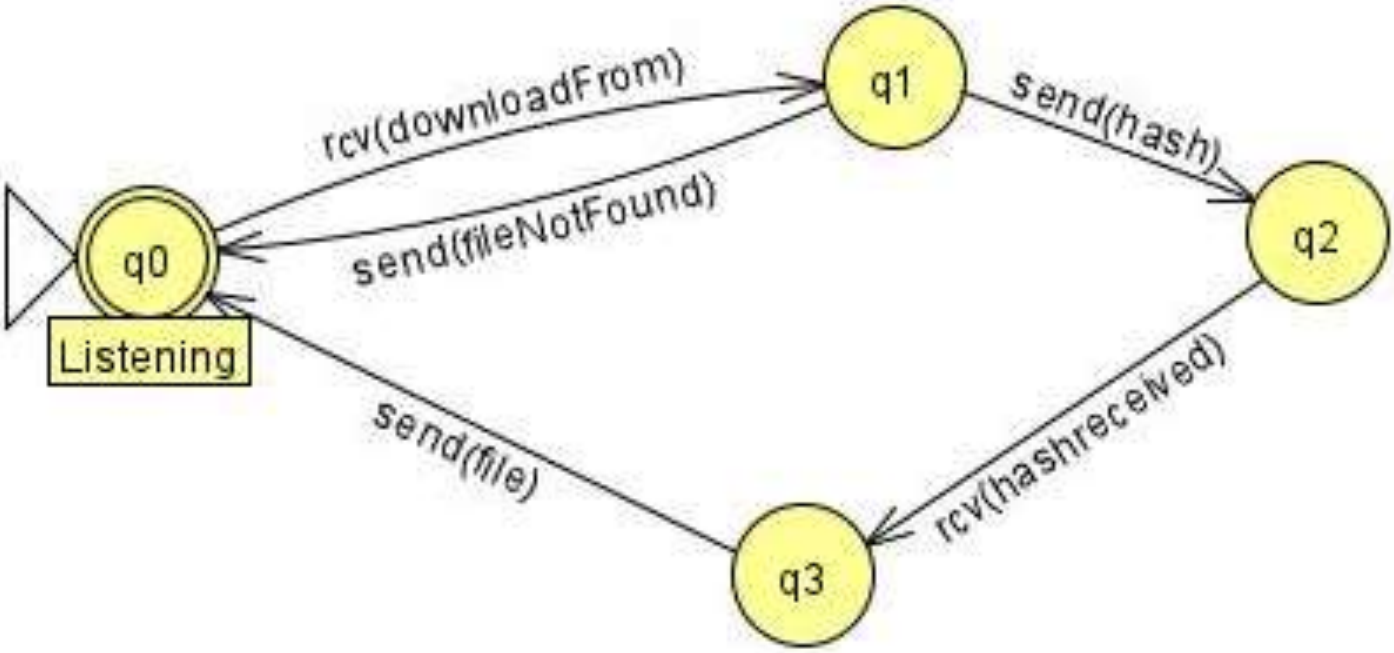
Ejemplo:

Opcode (1 byte)
5

Autómata rol cliente de ficheros:



Autómata rol servidor de ficheros:



Mejoras implementadas

A continuación, se muestra un listado de las mejoras implementadas en la aplicación.

- **FGServe puerto variable (+0,5)**: si al iniciar el servidor el puerto de escucha predeterminado (10000) no está disponible, el servidor puede utilizar otros números de puerto que sí están disponibles. Esta mejora se ha implementado modificando el constructor de la clase NFServerSimple que modela dicho servidor.

```
try{
      InetSocketAddress serverSocketAddress =
                        new InetSocketAddress(PORT);
      serverSocket = new ServerSocket();
      serverSocket.bind(serverSocketAddress);
   }
   catch(BindException be) {
        InetSocketAddress serverSocketAddress =
                          new InetSocketAddress(0);
        serverSocket = new ServerSocket();
        serverSocket.bind(serverSocketAddress);
   }
```

Si el puerto está usado, el constructor del socket manda una *Bind Exception*. De esta manera, se atrapa la excepción y se vuelve a crear un socket con un puerto efímero (elegido por el sistema operativo).

- **BGServe Secuencial (+1) + Multihilo (+0,5)**: se inicia un servidor en segundo plano, de manera que el host que esté ejecutando dicho servidor pueda seguir introduciendo comandos (*userlist, downloadfrom…*) sin necesidad de pararlo. Esta mejora se ha implementado usando hilos: al crear el servidor se llama a un hilo para que quede a la escucha de nuevas conexiones. De esta manera se puede seguir usando el programa como normalmente se hace. Otra mejora añadida es que cada vez que se establece una conexión, se trata con otro hilo de la clase NFServerThread. De esta manera, vamos a poder atender varios clientes a la vez.

- **BGServe puerto efímero (+0,5)**: de la misma manera que se ha implementado el puerto efímero para el FGServe, solo que esta vez no probamos primero con un puerto específico.

- **Downloadfrom por nickname (+1):** para implementar esta mejora, cuando se inicia un servidor en segundo plano, el cliente manda un mensaje al directorio para registrar el puerto donde está escuchando. De esta manera, cuando otro cliente pregunta por la IP asociada al nickname enviado, el directorio puede consultar en un HashMap si la session key de ese nickname está asociada a una IP. Si lo está, el directorio devuelve la IP al cliente y este realiza el downloadfrom. Si no está, se informa al cliente que no se encontró ninguna IP asociada.

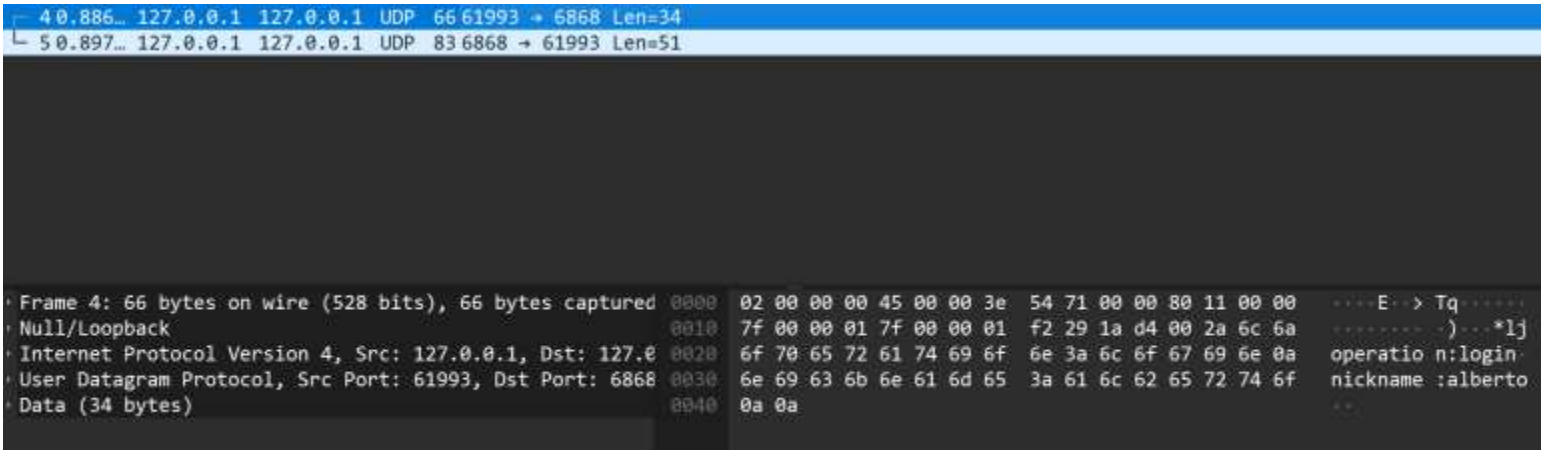
- **Userlist ampliado con servidores (+0,5):** cuando se ejecuta el comando userlist y se obtiene una lista con los nicknames de los usuarios registrados, por cada usuario registrado, se le pregunta al directorio si se puede resolver ese nickname como una IP. Si el directorio devuelve una IP (si se puede resolver), significa que ese nickname está registrado como un servidor y, por tanto, se imprime en pantalla como servidor.

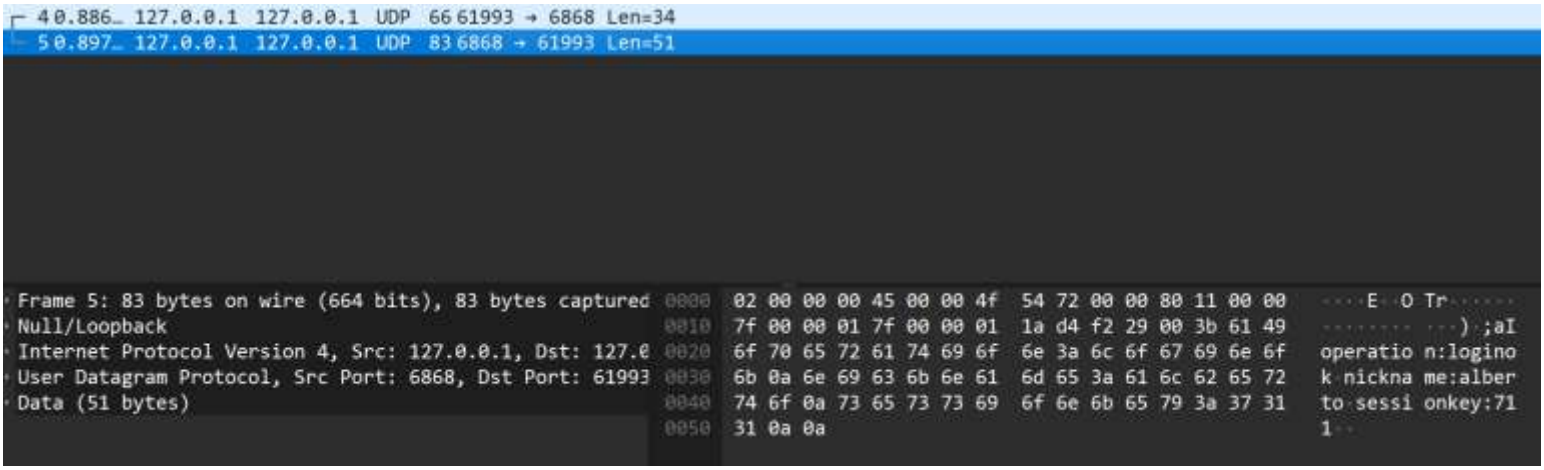
- **Stopserver (+0,5):** para parar el servidor en segundo plano, se usó una variable auxiliar llamada *stopServer*. Esta variable se cambia cuando se hace la llamada del comando, además de cerrar el socket. Cuando se produce el timeout de 1 segundo, el servidor atrapa la excepción *Socket Timeout Exception* y comprueba que la variable esté en *true*. Si es el caso, entonces sale del bucle que espera y trata las conexiones. Después de realizar esto, el cliente envía un mensaje para eliminarse del registro de servidores que lleva el directorio. El directorio actúa de la misma manera que cuando se registra un servidor, pero eliminándolo.

Capturas de Wireshark

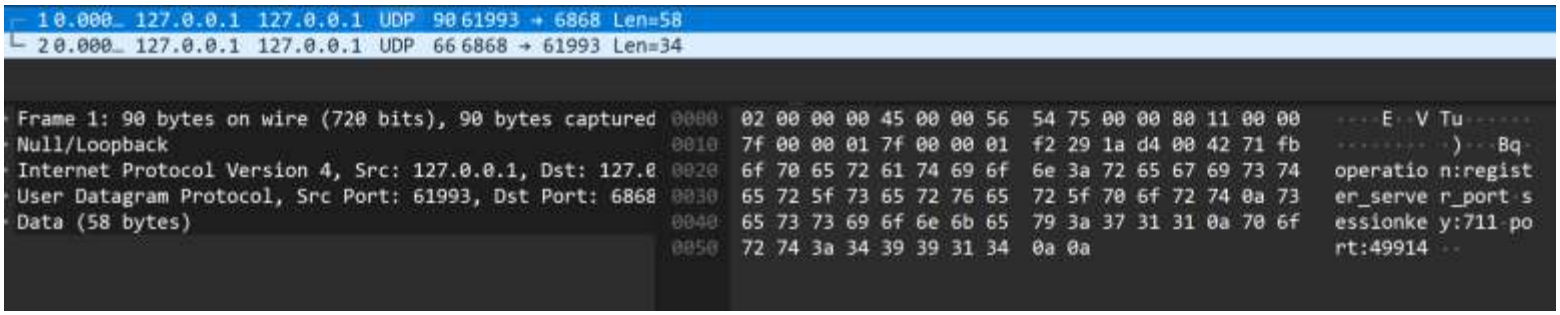
A continuación, se muestran una serie de capturas de pantalla para observar el funcionamiento de los mensajes.

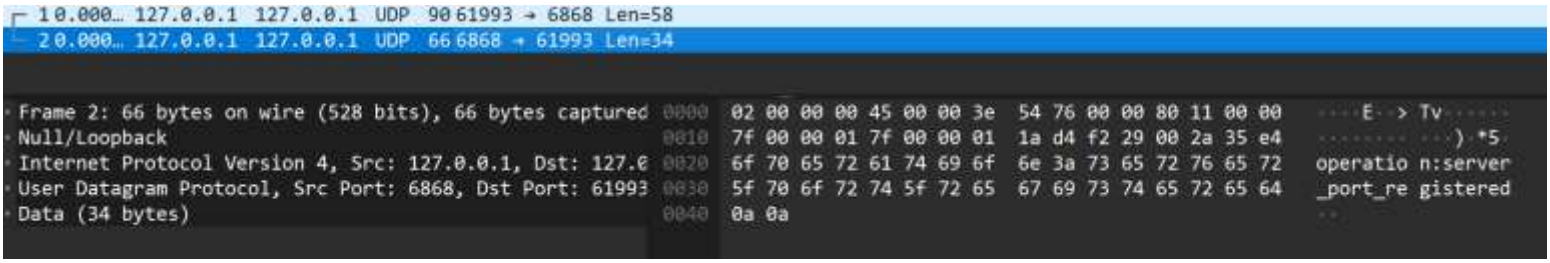
Comando login:



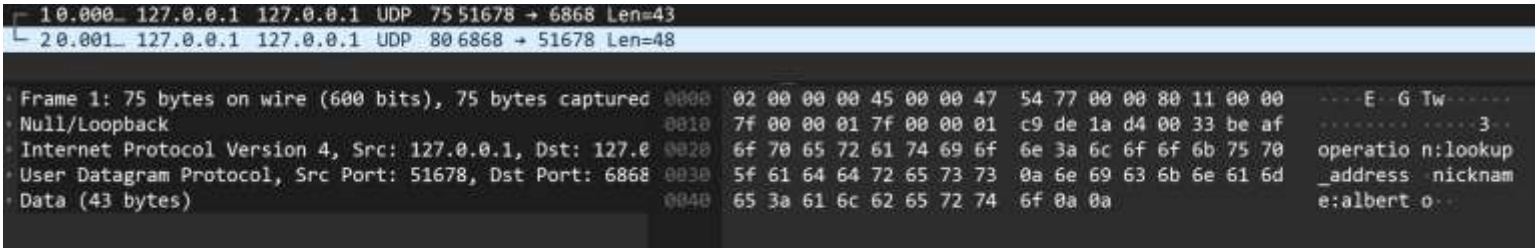


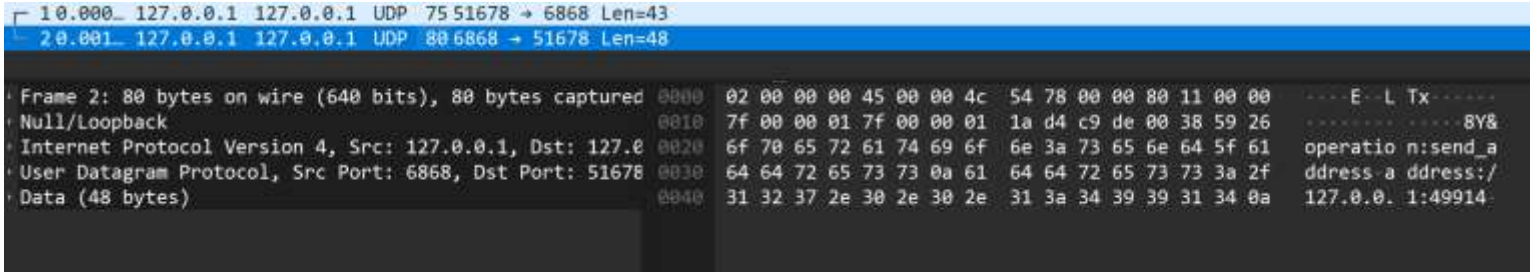
Comando bgserve:



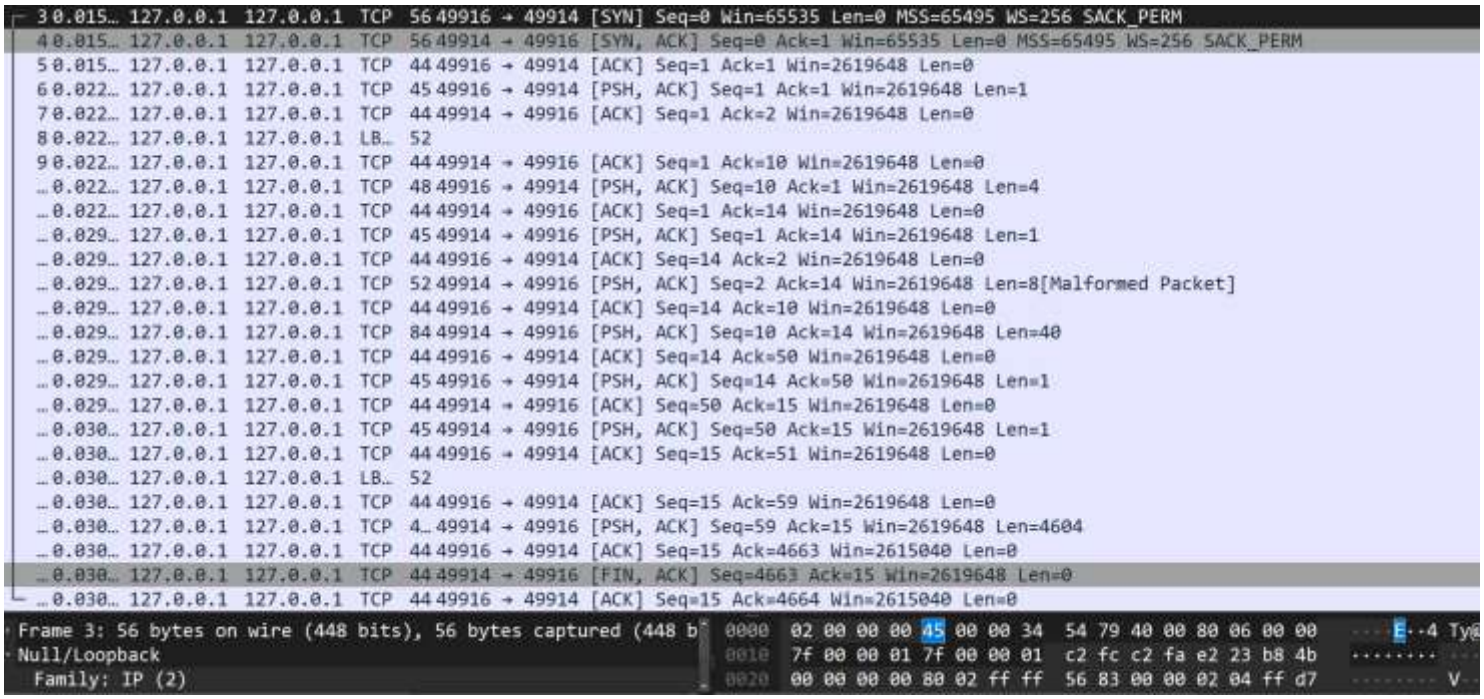


Comando downloadfrom nickname:

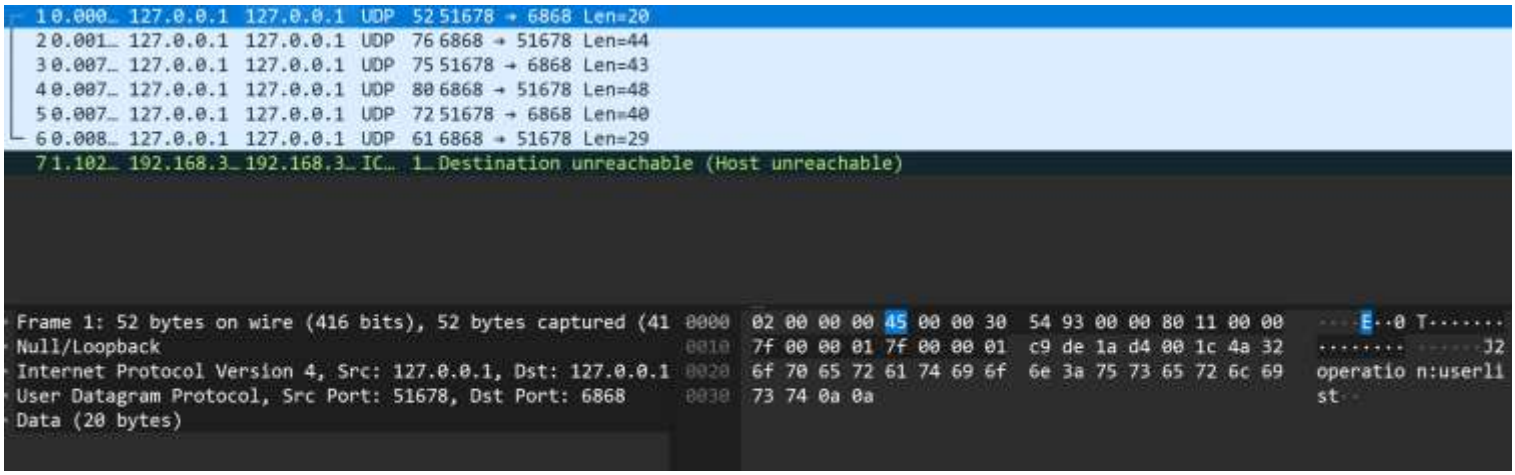


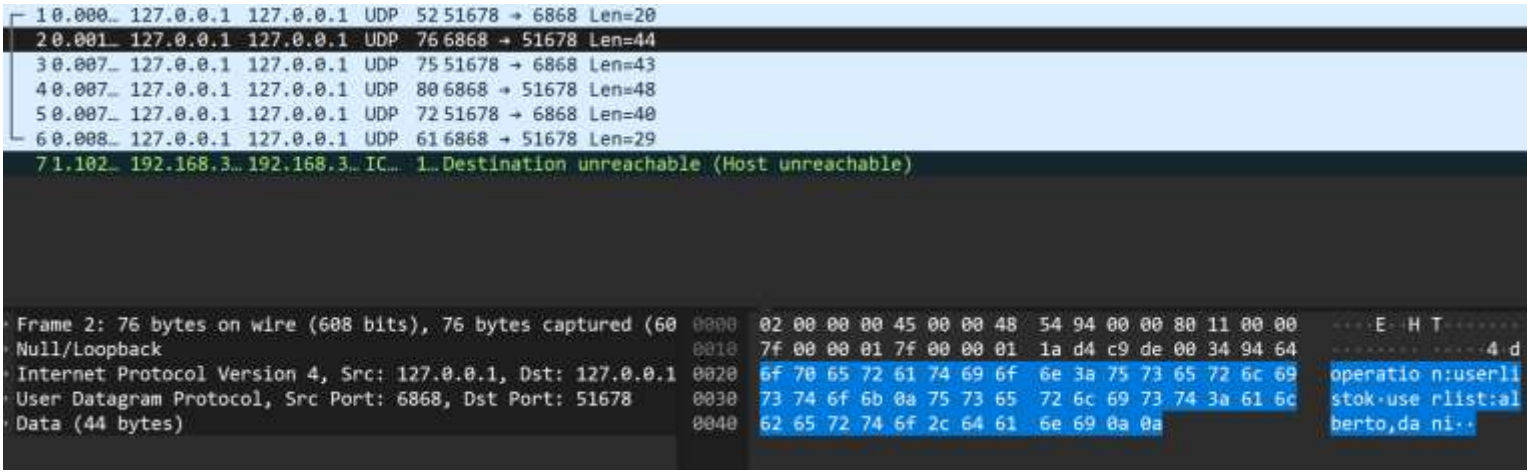


Al realizar una descarga, se genera todo este tráfico TCP:

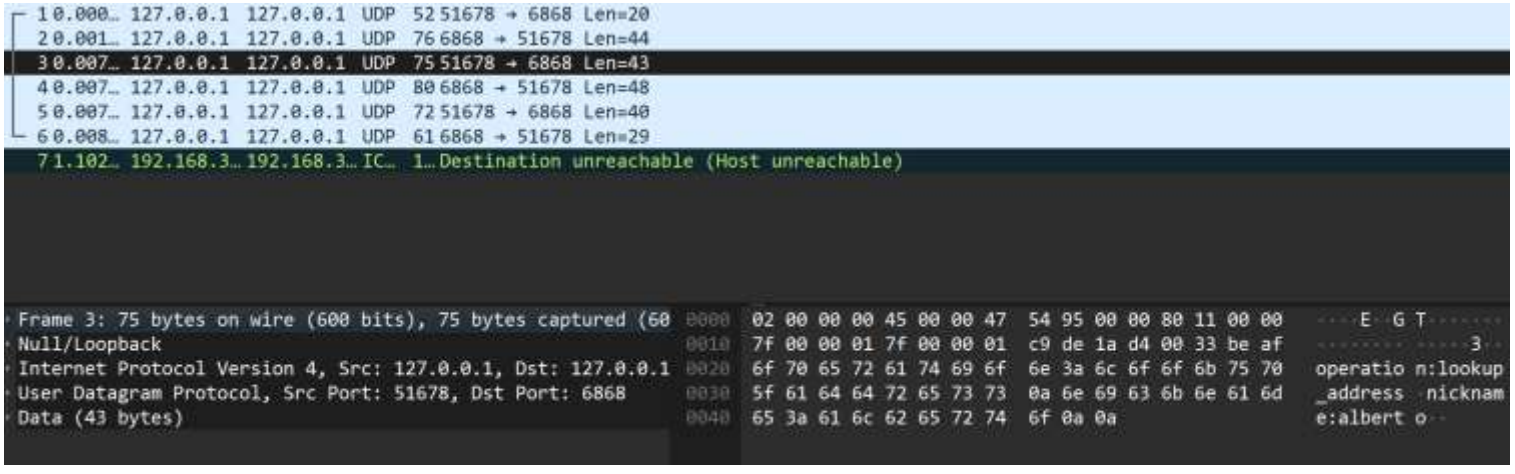


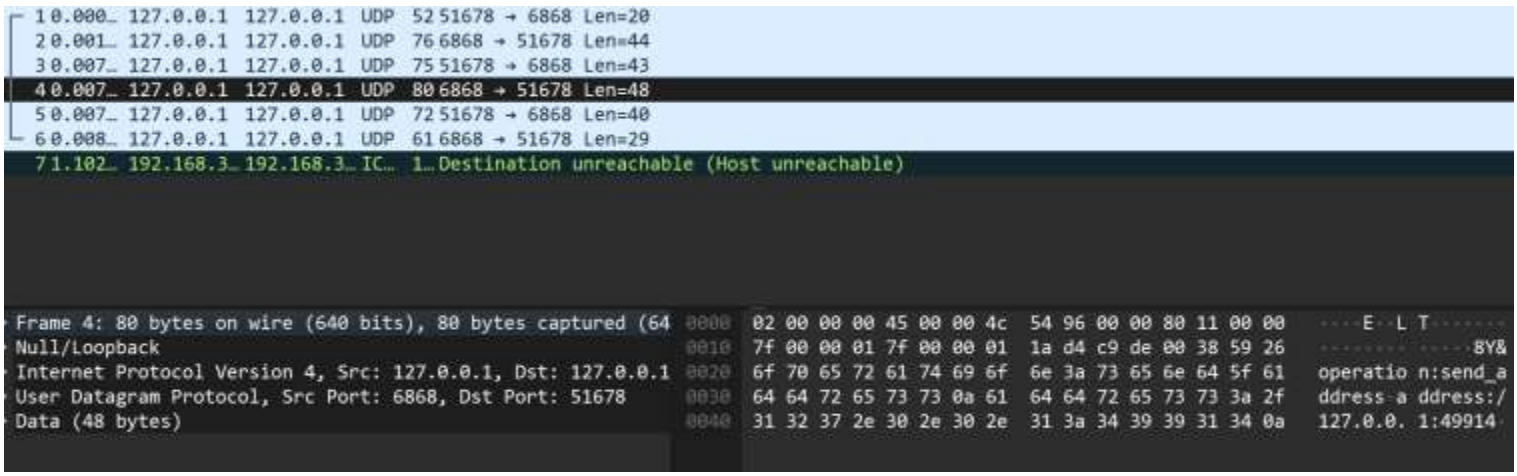
Comando userlist:

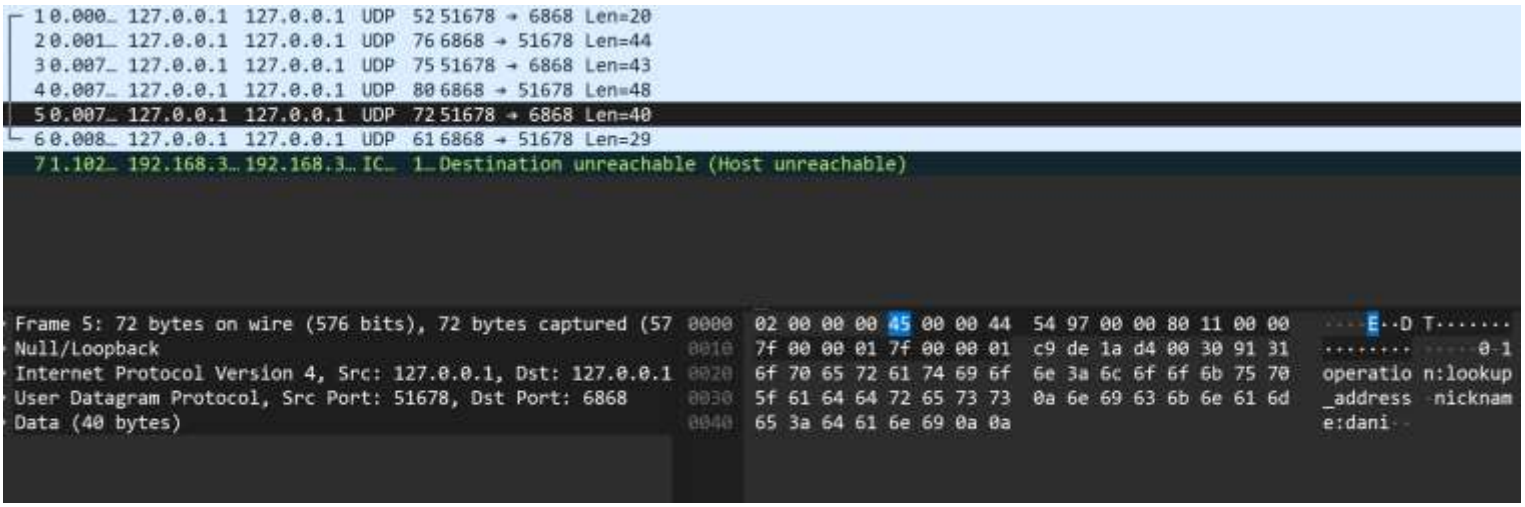


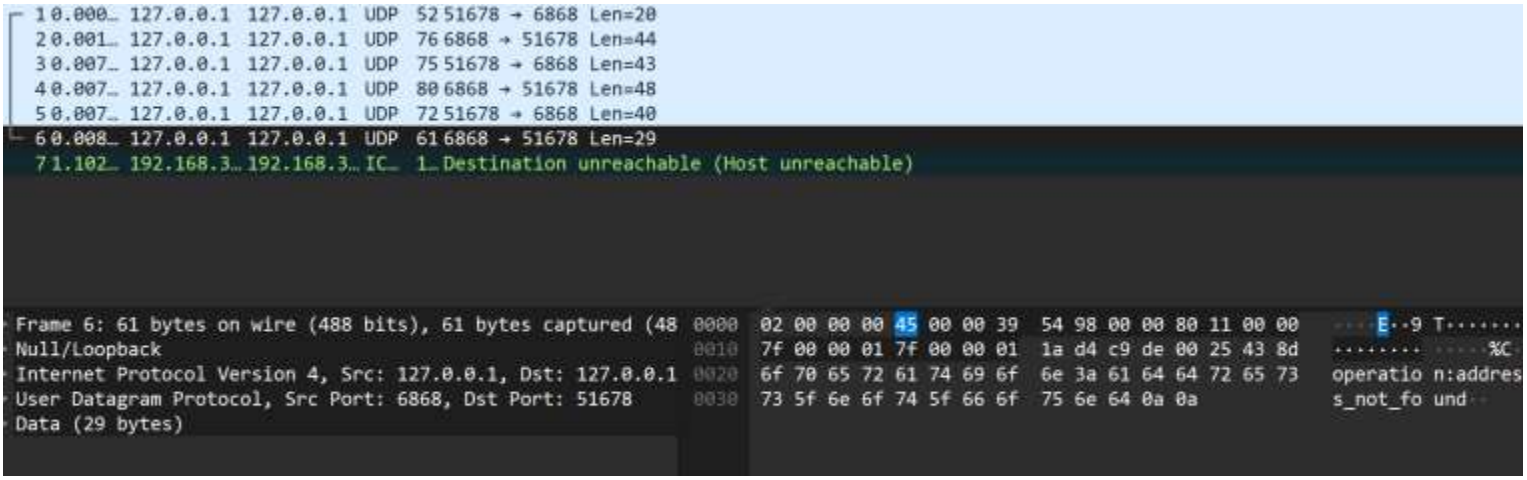


Para consultar qué usuarios son servidor y así imprimirlos como tal:






1 0.000… 127.0.0.1 127.0.0.1 UDP 52 51678 → 6868 Len=20
2 0.001… 127.0.0.1 127.0.0.1 UDP 76 6868 → 51678 Len=44
3 0.007… 127.0.0.1 127.0.0.1 UDP 75 51678 → 6868 Len=43
4 0.007… 127.0.0.1 127.0.0.1 UDP 80 6868 → 51678 Len=48
5 0.007… 127.0.0.1 127.0.0.1 UDP 72 51678 → 6868 Len=40
6 0.008… 127.0.0.1 127.0.0.1 UDP 61 6868 → 51678 Len=29
7 1.102… 192.168.3… 192.168.3… IC… 1… Destination unreachable (Host unreachable)
Frame 5: 72 bytes on wire (576 bits), 72 bytes captured (57
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 51678, Dst Port: 6868
Data (40 bytes)
0000 02 00 00 00 45 00 00 44 54 97 00 00 80 11 00 00 ····E··D T·····
0010 7f 00 00 01 7f 00 00 01 c9 de 1a d4 00 30 91 31 ····· ····0·1
0020 6f 70 65 72 61 74 69 6f 6e 3a 6c 6f 6f 6b 75 70 operatio n:lookup
0030 5f 61 64 64 72 65 73 73 0a 6e 69 63 6b 6e 61 6d _address ·nicknam
0040 65 3a 64 61 6e 69 0a 0a e:dani··


1 0.000… 127.0.0.1 127.0.0.1 UDP 52 51678 → 6868 Len=20
2 0.001… 127.0.0.1 127.0.0.1 UDP 76 6868 → 51678 Len=44
3 0.007… 127.0.0.1 127.0.0.1 UDP 75 51678 → 6868 Len=43
4 0.007… 127.0.0.1 127.0.0.1 UDP 80 6868 → 51678 Len=48
5 0.007… 127.0.0.1 127.0.0.1 UDP 72 51678 → 6868 Len=40
6 0.008… 127.0.0.1 127.0.0.1 UDP 61 6868 → 51678 Len=29
7 1.102… 192.168.3… 192.168.3… IC… 1… Destination unreachable (Host unreachable)
Frame 6: 61 bytes on wire (488 bits), 61 bytes captured (48
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 6868, Dst Port: 51678
Data (29 bytes)
0000 02 00 00 00 45 00 00 39 54 98 00 00 80 11 00 00 ····E··9 T·····
0010 7f 00 00 01 7f 00 00 01 1a d4 c9 de 00 25 43 8d ····· ····%C·
0020 6f 70 65 72 61 74 69 6f 6e 3a 61 64 64 72 65 73 operatio n:addres
0030 73 5f 6e 6f 74 5f 66 6f 75 6e 64 0a 0a s_not_fo und··

Conclusiones

Este proyecto ha sido el más completo de lo que llevamos de carrera. Cuando hemos usado aplicaciones de transferencia de archivos, transferencia de mensajes, juegos online… siempre nos hemos preguntado cómo funcionan las comunicaciones a nivel de software (y a nivel físico, pero esa pregunta nos la ha resuelto la teoría de esta asignatura).

Al principio, nos era difícil pensar cómo íbamos a implementar mensajes y protocolos de comunicación (debido a la complejidad del código, estando acostumbrados a proyectos de uno o dos ficheros), pero conforme íbamos realizando los boletines de la práctica, íbamos entendiendo el código cada vez más y encajando las piezas del puzle.

Para nosotros ha sido un gran aprendizaje realizar este proyecto que, sin duda, nos ha servido de mucho para nuestro desarrollo como profesionales. Ahora sabemos lo que es trabajar en un proyecto amplio, con hilos de por medio, y además con conexiones por sockets. Este proyecto será de los que, una vez acabemos la carrera, nos enorgullecerá enseñar a la gente para mostrar lo que realizábamos en ella.