Diseño de la implementación del proyecto de NanoFiles
Redes de comunicaciones
Convocatoria junio 2023

Sergio Ros Liarte
José maría López Serrano

Índice

Introducción

En este documento se presentan los autómatas que modelan el comportamiento de nuestra implementación del proyecto de NanoFiles para la asignatura de Redes de comunicaciones en la convocatoria de mayo de 2023. También incluimos especificación de todos los tipos de mensaje utilizados por el programa Directory y NanoFiles para comunicarse entre ellos y los mensajes de NanoFiles para comunicación entre pares. Las mejoras adicionales que hemos realizado al proyecto son:

- Explorar ficheros remotos mediante nickname (browse <nick>) (1 punto)
- Ampliar userlist con información sobre servidores (0.5 puntos)
- Mantener actualizados servidores disponibles (0.5 puntos)
- Consultar ficheros disponibles en el explorador (queryfiles) (0.5 puntos)
- Detener servidor en primer plano (fgstop) (0.5 puntos)

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) 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: Longitud-Valor

Opcode (1 byte)	Longitud (4 bytes)	Valor (<Longitud> bytes)

Formato: Longitud-Valor+int

Opcode (1 byte)	Longitud (4 bytes)	Valor (<Longitud> bytes)	int (4 bytes)

Formato: IP:PORT

Opcode (1 byte)	IP (4 bytes)				PORT (4 bytes)

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:

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)
2	0

Mensaje: **Register (opcode = 6)**
Formato: Longitud-Valor
Sentido de la comunicación: Cliente→ Directorio
Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para solicitar registrar un nick y obtener una confirmación. El valor asignado al opcode es 6.

Ejemplo: un cliente solicita registrar el nick "juanp", de longitud 5.

Opcode (1 byte)	Longitud (4 bytes)	Valor (<Longitud> bytes)
6	4	juan

Mensaje: **RegisterOK (opcode = 7)**
Formato: Control
Sentido de la comunicación: Directorio→ Cliente
Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles para confirmar el registro del nick del usuario en el servidor. El valor asignado al opcode es 7.

Ejemplo:

Opcode (1 byte)
7

Mensaje: **RegisterNotNOTOK (opcode = 8)**
Formato: Control
Sentido de la comunicación: Directorio→ Cliente
Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles para decirle que su nick no se ha podido registrar, normalmente porque ya está registrado. El valor asignado al opcode es 8.

Ejemplo:

Opcode (1 byte)
8

Mensaje: **Userlist(request) (opcode = 13)**
Formato: Control
Sentido de la comunicación: Cliente→ Directorio
Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para solicitar la lista de usuarios que este tiene almacenada. El valor asignado al opcode es 13.

Ejemplo:

Opcode (1 byte)
13

Mensaje: **Userlist(response) (opcode = 14)**
Formato: Longitud-Valor
Sentido de la comunicación: Cliente→ Directorio
Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para devolver al cliente la lista de usuarios, ya formateada preparada para imprimirla por pantalla. Además contiene información sobre los usuarios que están sirviendo ficheros. El valor asignado al opcode es 14.

Ejemplo:

Opcode	Longitud	Valor

(1 byte)	(4 bytes)	(<Longitud> bytes)
14	22	nicks: juan - S; pepe;

Mensaje: **LookupUsername(opcode = 3)**
Formato: Longitud-valor
Sentido de la comunicación: Cliente→Directorio
Descripción: Este mensaje lo envía un cliente a el dircectorio de NanoFiles para solicitar la IP y el puerto asignadas al usuario que ha solicitado el cliente. El valor asignado al opcode es 3.

Ejemplo:

Opcode (1 byte)	Longitud (4 bytes)	Valor (<Longitud> bytes)
3	5	juanp

Mensaje: **UserNameNotFound (opcode = 5)**
Formato: Control
Sentido de la comunicación: Directorio→ Cliente
Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles para informar que el usuario solicitado por el cliente no existe. El valor asignado al opcode es 5.

Ejemplo:

Opcode (1 byte)
5

Mensaje: **UsernameFound (opcode = 4)**
Formato: IP-Port
Sentido de la comunicación: Directorio→Cliente
Descripción: Este mensaje lo envía el directorio de NanoFiles a un cliente para devolver al cliente la IP y el puerto asignadas al usuario que ha solicitado el cliente. El valor asignado al opcode es 4.

Ejemplo:

Opcode (1 byte)	Longitud (4 bytes)				Valor (4 bytes)
4	192	168	1	10	9657

Mensaje: **Serving (opcode = 9)**
Formato: Longitud-Valor+int
Sentido de la comunicación: Directorio→Cliente
Descripción: Este mensaje lo envía un cliente de NanoFiles a al directorio para

indicarle que quiere empezar a servir ficheros con el nick indicado (valor) y que escucha en el puerto indicado (int).

Ejemplo:

Opcode (1 byte)	Longitud (4 bytes)	Valor (4 bytes)	Int (4 bytes)
9	4	juan	10000

Mensaje: **ServingOK (opcode = 10)**

Formato: Control

Sentido de la comunicación: Directorio→ Cliente

Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles para informar de que ha aceptado su petición de servir ficheros y ha guardado su nick, su ip y su puerto de escucha en su registro de servidores para facilitarselo a clientes que pregunten por esa IP:puerto.

Ejemplo:

Opcode (1 byte)
10

Mensaje: **AlreadyServing (opcode = 11)**

Formato: Control

Sentido de la comunicación: Directorio→ Cliente

Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles para informar de que ha rechazado su solicitud de servir ficheros porque ya está sirviendo en esa IP y ese puerto en concreto.

Ejemplo:

Opcode (1 byte)
11

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

Formato: Control

Sentido de la comunicación: Directorio→ Cliente

Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles para informar de que ha rechazado su solicitud de servir ficheros porque ya hay alguien sirviendo con ese nick.

Ejemplo:

Opcode (1 byte)
12

En realidad ServingNOTOK y AlreadyServing en una ejecución normal de nuestra implementación nunca van a ser utilizados porque no tenemos un servidor concurrente que pueda comunicarle al directorio que escucha varias veces en el mismo puerto (Already), y porque el mensaje de Serving se realiza con el nick que tiene guardado NanoFiles del registro en el directorio y el directorio no admite nicks repetidos, así que nunca un servidor va a intentar registrarse con el nick de otra persona. Aún así decidimos

incluir ambos para que no se de por sentado que siempre se aceptará la solicitud de servir ficheros (aunque en nuestra implementación sea así).

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

Formato: Longitud-valor

Sentido de la comunicación: Cliente→Directorio

Descripción: Este mensaje lo envía un servidor al directorio para indicarle que ha dejado de servir ficheros. Solo le comunica el nick porque estos son únicos y como solo tenemos implementado el servidor en primer plano nunca un servidor estará escuchando en varios puertos.

Ejemplo:

Opcode (1 byte)	Longitud (4 bytes)	Valor (<Longitud> bytes)
15	4	juan

Mensaje: **ServingStop(response) (opcode = 16)**

Formato: Control

Sentido de la comunicación: Directorio→ Cliente

Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles para informar de que ha recibido su comunicación de dejar de servir y ha eliminado la entrada de su tabla de servidores.

Ejemplo:

Opcode (1 byte)
16

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

Formato: Longitud-valor

Sentido de la comunicación: Cliente→Directorio

Descripción: Este mensaje lo envía un servidor al directorio para indicarle que quiere dar de baja su nick del directorio para cerrar el programa.

Ejemplo:

Opcode (1 byte)	Longitud (4 bytes)	Valor (<Longitud> bytes)
17	4	juan

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

Formato: Control

Sentido de la comunicación: Directorio→ Cliente

Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles para informar de que ha recibido su petición de dar de baja su nick y lo ha hecho, dejando libre el nick.

Ejemplo:

Opcode

(1 byte)
18

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: FileCode

operation: xxxxxxxx\n
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Formato: Control
operation: xxxxxxx\n
\n

Formato: FileSize
operation: xxxxxxx\n
size: xxx\n
\n

Formato: Query
operation: xxxxxx\n
files: xxxxxxxxxx;xxxxxxxxxxxxx;xxxxxxxxxxx;hash\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: **FileSize**
Formato: FileSize
Sentido de la comunicación: Servidor de ficheros→ Cliente
Descripción: Este mensaje lo envía el servidor de ficheros al cliente para indicarle cuantos paquetes de código va a enviarle del fichero que ha solicitado antes el cliente.
Ejemplo:

operation: filesize\n
size: 7\n
\n

Mensaje: **FileCode**
Formato: Code
Sentido de la comunicación: Servidor de ficheros→ Cliente
Descripción: Este mensaje lo envía el servidor de ficheros al cliente con parte (o todo) del código fuente del fichero que ha solicitado el cliente. Este mensaje solo tiene un campo y el resto es código porque creíamos que otro campo solo nos hacía perder capacidad por mensaje.
Ejemplo:

operation: code\n
ax´´´121489'adheoigflajwgd_.,m–,-,-,-.´+``ççñ+asuwupd

(cadena codificada en base 64 con código fuente binario)

Mensaje: **QueryFiles**
Formato: Control
Sentido de la comunicación: Cliente→ Servidor de ficheros
Descripción: Este mensaje lo envía el cliente al servidor de ficheros para solicitar información sobre los ficheros que este sirve.
Ejemplo:

operation: queryfiles\n
\n

Mensaje: **QueryResponse**
Formato: Query
Sentido de la comunicación: Servidor de ficheros→ Cliente
Descripción: Este mensaje lo envía el servidor de ficheros al cliente con información sobre los ficheros que está sirviendo.
Ejemplo:

operation: queryresponse\n
files: 87071b18127a076887613fa2b38511ec; 1984.pdf; 12;\n
\n

Mensaje: **QueryFiles**

Formato: Control

Sentido de la comunicación: Cliente→ Servidor de ficheros

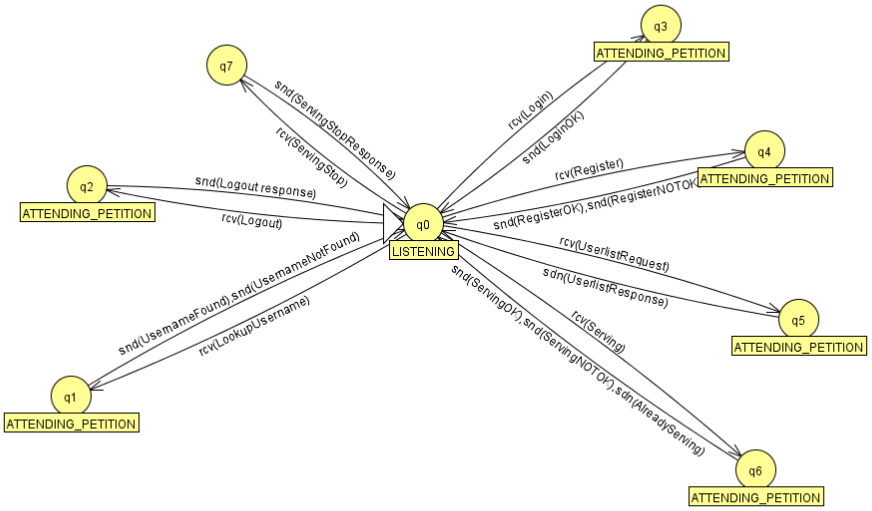
Descripción: Este mensaje lo envía el cliente al servidor de ficheros para cerrar la comunicación. Cuando el servidor lo recibe le envía uno igual de confirmación al cliente.

Ejemplo:

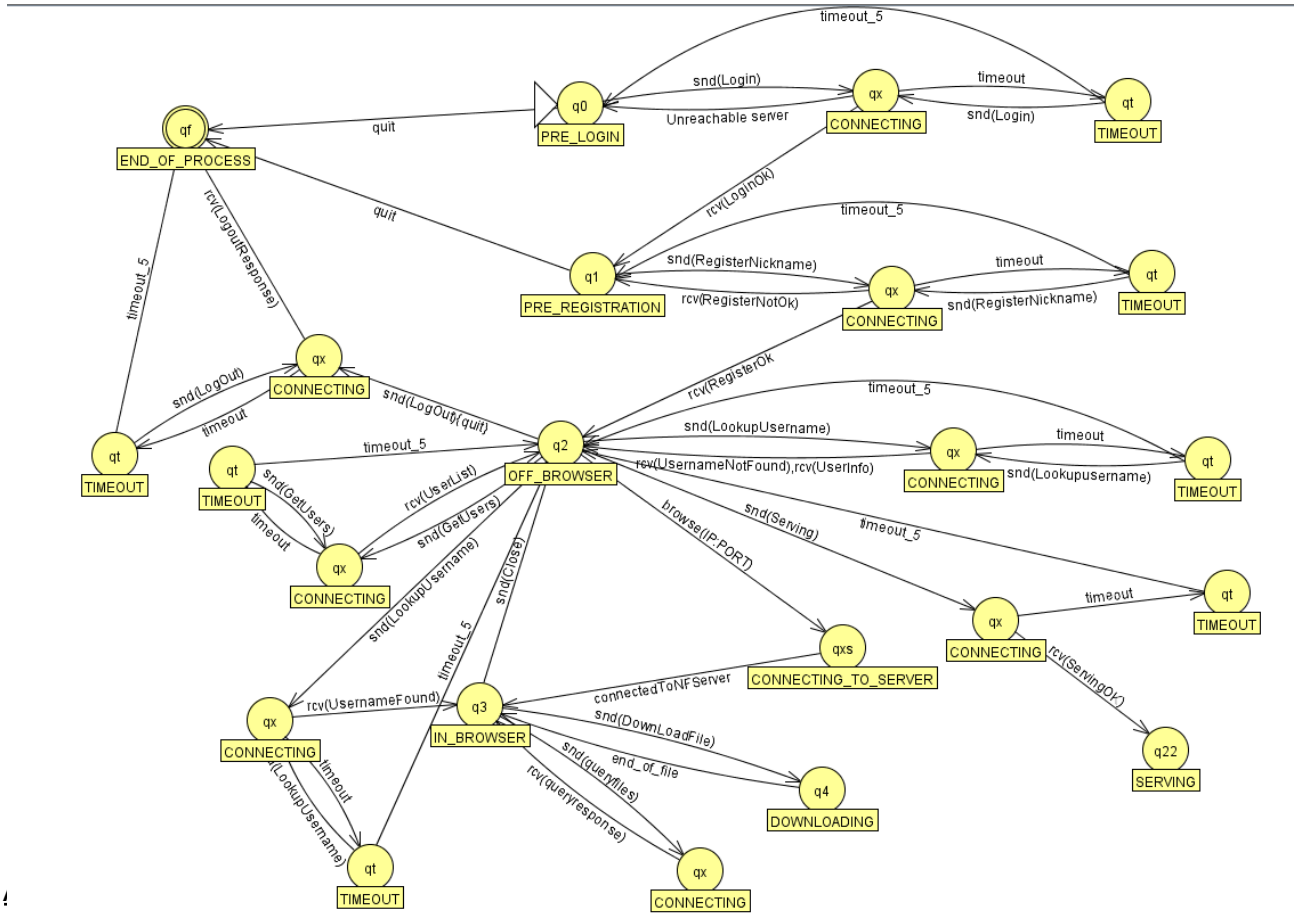
operation: close\n
\n

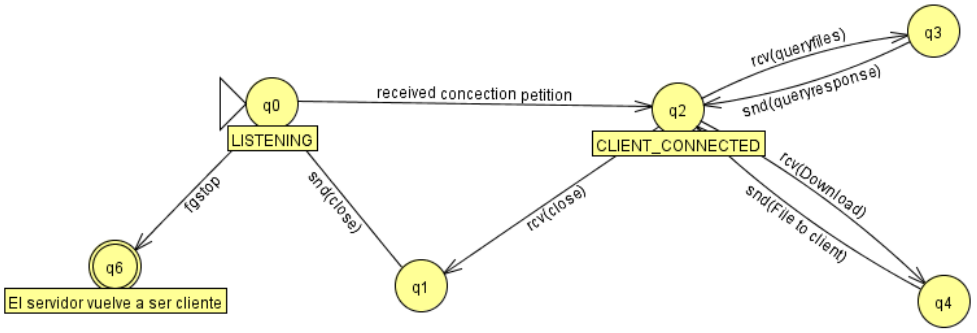
Autómatas de protocolo

Autómata Servidor de Directorio



Autómata Cliente (unificado TCP/UDP)





Ejemplo de intercambio de mensajes

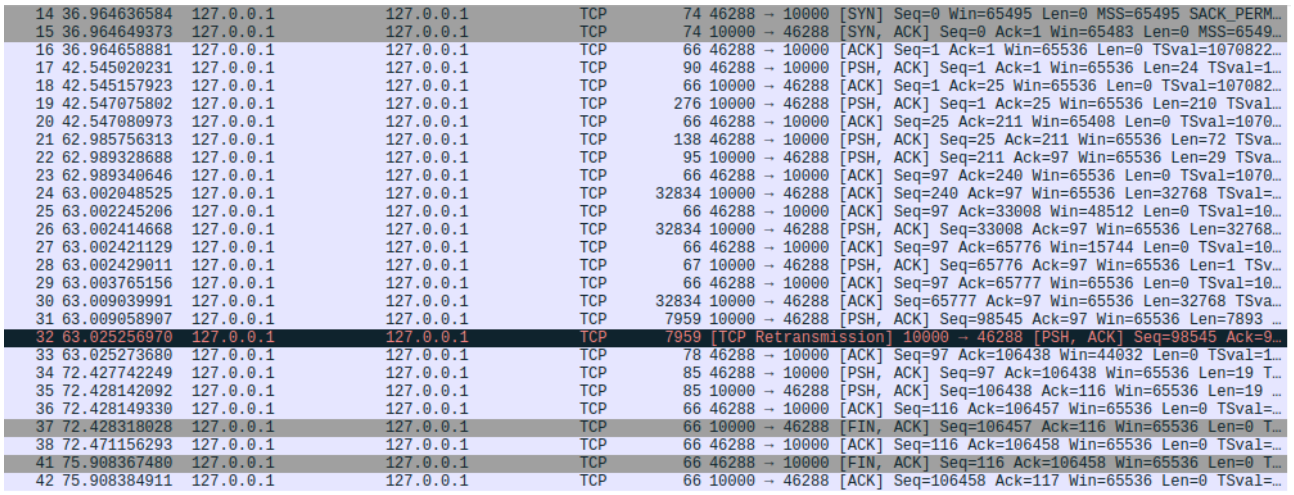
Conversación entre 1 directorio (D) y dos clientes NF1 y NF2. En esta conversación, primero NF1 se registra en el servidor con el nombre juan y empieza a servir ficheros. NF2 después intenta también registrarse como juan pero no puede porque ya está en uso. Se registra como luis y pide la lista de usuarios. Después pide al directorio la IP y el puerto del usuario juan y se conecta a su servidor TCP. Por último le hace una query para ver los ficheros que NF1 tiene. este le manda la respuesta y acto seguido NF2 le envía un mensaje de cierre. NF1 contesta con el mismo mensaje y se acaba la comunicación. Por último ambos se dan de baja del directorio.

D - escuchando…
NF1 -> D: Login: 1
D -> NF1: LoginOK: 2(0)
NF1 -> D: Register: 6(4)(juan)
D -> NF1: RegisterOK: 7
NF1 -> D: Serving: 9(4)(juan)(10000)
D -> NF1: ServingOK: 10
NF1 - escuchando…
NF2 -> D: Login: 1
D -> NF2: LoginOK: 2(1)
NF2 -> D: Register: 6(4)(juan)
D -> NF2: RegisterNOTOK: 7
NF2 -> D: Register: 6(4)(luis)

D -> NF2: RegisterOK: 7
NF2 -> D: Userlist: 13
D -> NF2: Userlist: 14(22)(nicks: juan - S; luis;)
NF2 -> D: LookupUsername: 3(4)(juan)
D -> NF2: UsernameFound: 4(192.168.1.1)(10000)
NF2 -> NF1: Connect to peer… IN BROWSER
NF2 -> NF1: queryfiles: operation:queryfiles\n\n
NF1 -> NF2: queryresponse: operation:queryresponse\nfiles:56839haywdkfxc; memory.pdf; 60;\nmfdskfh23489zxd;userdata;34;\n\n
NF2 -> NF1: close: operation:close\n\n
NF1 -> NF2: close: operation:close\n\n
NF1 -> D: ServingStop: 15(4)(juan)
D -> NF1: ServingStopOK: 16
NF1 -> D: Logout: 17(4)(juan)
D -> NF1: LogoutResponse: 18
NF2 -> D: Logout: 17(4)(luis)
D -> NF2: LogoutResponse: 18

Capturas de wireshark

En este apartado se incluyen unas capturas realizadas con Wireshark donde se ve el flujo de mensajes TCP de una conexión entre peers para la descarga de un fichero de más de 64Kb (más de un paquete de código).



```
▸ Frame 17: 90 bytes on wire (720 bits), 90 bytes captured (720 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: 46288, Dst Port: 10000, Seq: 1, Ack: 1, Len: 24
▾ Data (24 bytes)
    Data: 00166f7065726174696f6e3a71756572796669 6c65730a0a
    [Length: 24]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  00 4c ba 2e 40 00 40 06  82 7b 7f 00 00 01 7f 00   ·L··@·@· ·{····
0020  00 01 b4 d0 27 10 e9 15  1d b4 88 6b 73 11 80 18   ····'··· ···ks···
0030  02 00 fe 40 00 00 01 01  08 0a 3f d3 8a 44 3f d3   ···@···· ··?··D?·
0040  74 77 00 16 6f 70 65 72  61 74 69 6f 6e 3a 71 75   tw··oper ation:qu
0050  65 72 79 66 69 6c 65 73  0a 0a                     eryfiles ··
```

```
▸ Frame 19: 276 bytes on wire (2208 bits), 276 bytes captured (2208 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: 10000, Dst Port: 46288, Seq: 1, Ack: 25, Len: 210
▾ Data (210 bytes)
    Data: 00d06f7065726174696f6e3a7175657279726573706f6e73…
    [Length: 210]
```

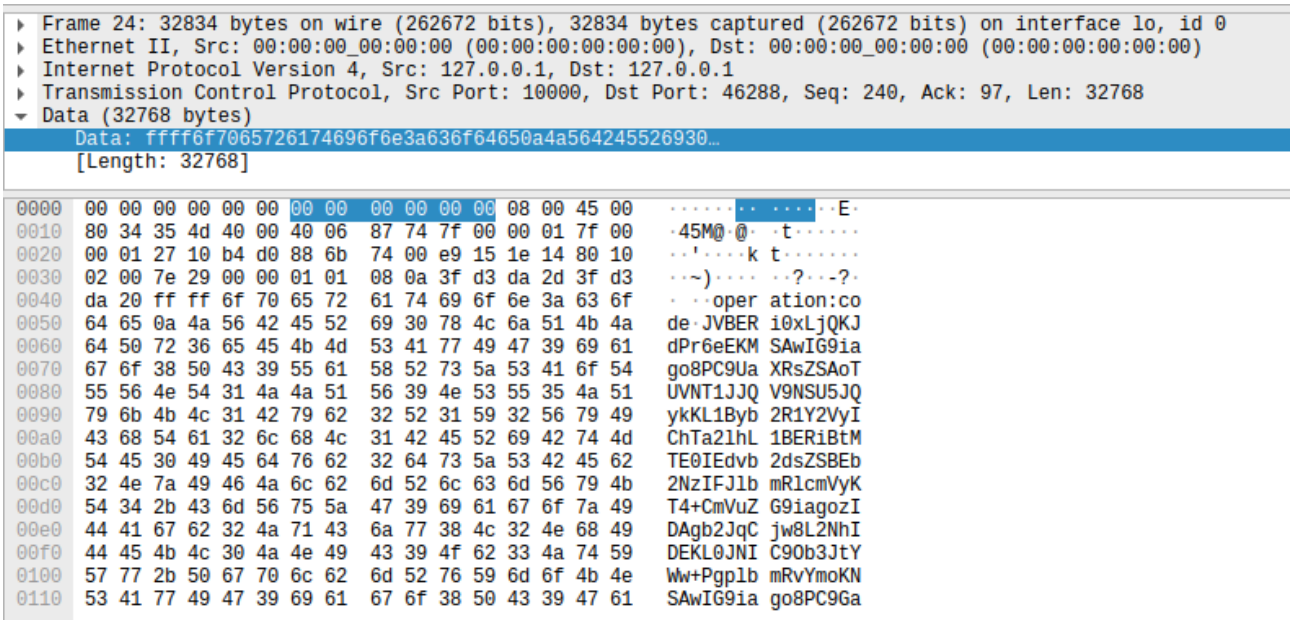
```
0040  8a 44 00 d0 6f 70 65 72  61 74 69 6f 6e 3a 71 75   ·D··oper ation:qu
0050  65 72 79 72 65 73 70 6f  6e 73 65 0a 66 69 6c 65   eryrespo nse·file
0060  73 3a 35 65 37 32 30 38  36 30 38 33 34 37 36 30   s:5e7208 60834760
0070  39 37 39 36 32 31 63 35  31 36 62 37 35 36 63 36   979621c5 16b756c6
0080  61 38 61 38 33 36 33 39  31 33 3b 4d 45 4d 4f 52   a8a83639 13;MEMOR
0090  49 41 5f 4d 49 4e 49 43  2e 70 64 66 3b 37 39 36   IA_MINIC .pdf;796
00a0  32 33 0a 36 61 61 64 65  31 34 30 61 32 34 65 32   23·6aade 140a24e2
```

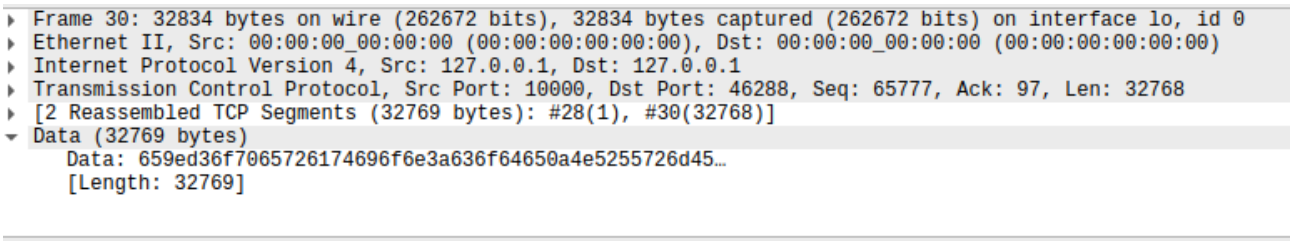
```
▸ Frame 21: 138 bytes on wire (1104 bits), 138 bytes captured (1104 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 46288, Dst Port: 10000, Seq: 25, Ack: 211, Len: 72
▾ Data (72 bytes)
    Data: 00466f7065726174696f6e3a646f776e6c6f61640a66696c…
    [Length: 72]
```

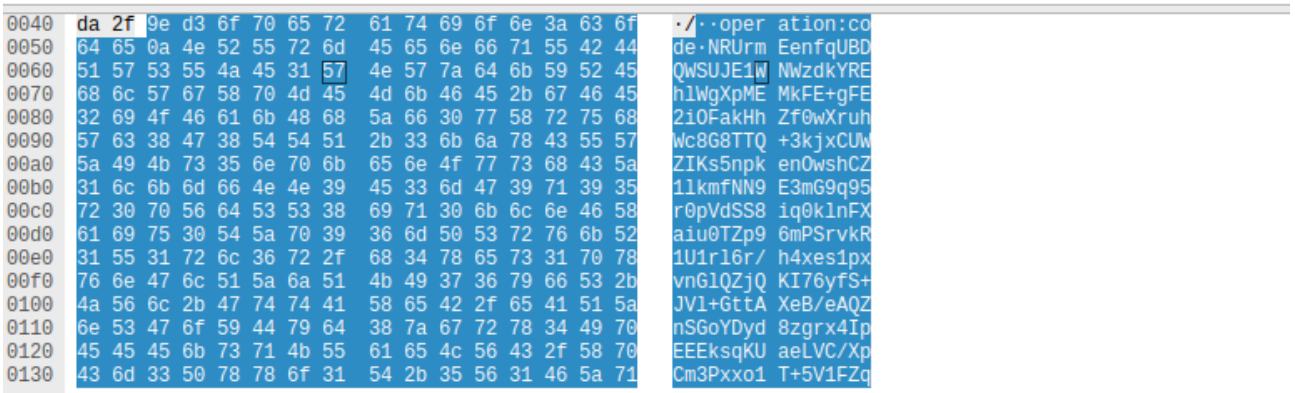
```
0020  00 01 b4 d0 27 10 e9 15  1d cc 88 6b 73 e3 80 18   ····'··· ···ks···
0030  02 00 fe 70 00 00 01 01  08 0a 3f d3 da 1c 3f d3   ···p···· ··?···?·
0040  8a 46 00 46 6f 70 65 72  61 74 69 6f 6e 3a 64 6f   ·F·Foper ation:do
0050  77 6e 6c 6f 61 64 0a 66  69 6c 65 68 61 73 68 3a   wnload·f ilehash:
0060  35 65 37 32 30 38 36 30  38 33 34 37 36 30 39 37   5e720860 83476097
0070  39 36 32 31 63 35 31 36  62 37 35 36 63 36 61 38   9621c516 b756c6a8
0080  61 38 33 36 33 39 31 33  0a 0a                     a8363913 ··
```

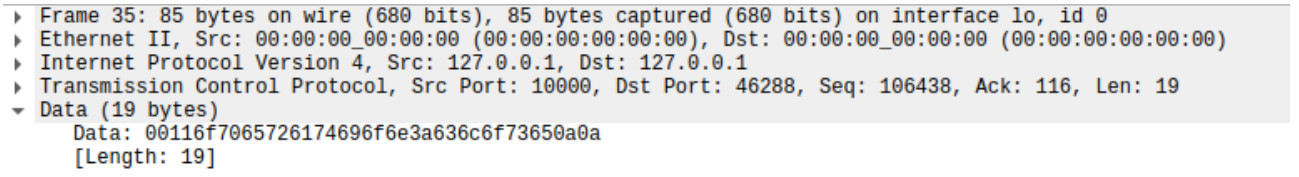
```
▸ Frame 22: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 10000, Dst Port: 46288, Seq: 211, Ack: 97, Len: 29
▾ Data (29 bytes)
    Data: 001b6f7065726174696f6e3a66696c6573697a650a73697a…
    [Length: 29]
```

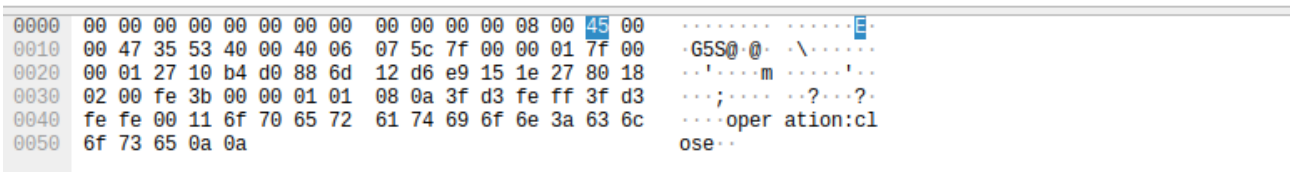
```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  00 51 35 4c 40 00 40 06  07 59 7f 00 00 01 7f 00   ·Q5L@·@· ·Y····
0020  00 01 27 10 b4 d0 88 6b  73 e3 e9 15 1e 14 80 18   ··'····k s·····
0030  02 00 fe 45 00 00 01 01  08 0a 3f d3 da 20 3f d3   ···E···· ··?·· ?·
0040  da 1c 00 1b 6f 70 65 72  61 74 69 6f 6e 3a 66 69   ····oper ation:fi
```

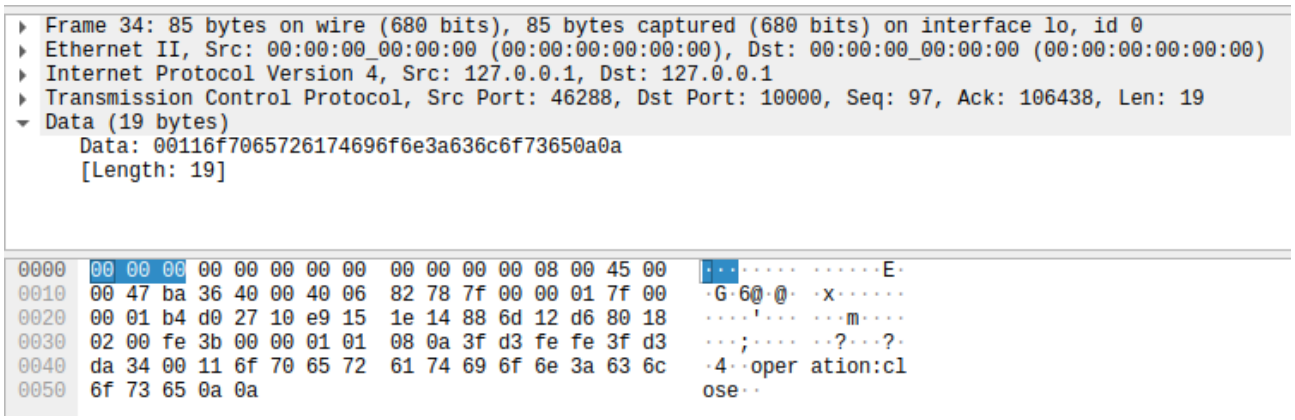

Frame 24: 32834 bytes on wire (262672 bits), 32834 bytes captured (262672 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 10000, Dst Port: 46288, Seq: 240, Ack: 97, Len: 32768
Data (32768 bytes)
Data: ffff6f7065726174696f6e3a636f64650a4a564245526930…
[Length: 32768]
0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00
0010 80 34 35 4d 40 00 40 06 87 74 7f 00 00 01 7f 00
0020 00 01 27 10 b4 d0 88 6b 74 00 e9 15 1e 14 80 10
0030 02 00 7e 29 00 00 01 01 08 0a 3f d3 da 2d 3f d3
0040 da 20 ff ff 6f 70 65 72 61 74 69 6f 6e 3a 63 6f
0050 64 65 0a 4a 56 42 45 52 69 30 78 4c 6a 51 4b 4a
0060 64 50 72 36 65 45 4b 4d 53 41 77 49 47 39 69 61
0070 67 6f 38 50 43 39 55 61 58 52 73 5a 53 41 6f 54
0080 55 56 4e 54 31 4a 4a 51 56 39 4e 53 55 35 4a 51
0090 79 6b 4b 4c 31 42 79 62 32 52 31 59 32 56 79 49
00a0 43 68 54 61 32 6c 68 4c 31 42 45 52 69 42 74 4d
00b0 54 45 30 49 45 64 76 62 32 64 73 5a 53 42 45 62
00c0 32 4e 7a 49 46 4a 6c 62 6d 52 6c 63 6d 56 79 4b
00d0 54 34 2b 43 6d 56 75 5a 47 39 69 61 67 6f 7a 49
00e0 44 41 67 62 32 4a 71 43 6a 77 38 4c 32 4e 68 49
00f0 44 45 4b 4c 30 4a 4e 49 43 39 4f 62 33 4a 74 59
0100 57 77 2b 50 67 70 6c 62 6d 52 76 59 6d 6f 4b 4e
0110 53 41 77 49 47 39 69 61 67 6f 38 50 43 39 47 61


Frame 30: 32834 bytes on wire (262672 bits), 32834 bytes captured (262672 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: 10000, Dst Port: 46288, Seq: 65777, Ack: 97, Len: 32768
[2 Reassembled TCP Segments (32769 bytes): #28(1), #30(32768)]
Data (32769 bytes)
Data: 659ed36f7065726174696f6e3a636f64650a4e5255726d45…
[Length: 32769]


0040 da 2f 9e d3 6f 70 65 72 61 74 69 6f 6e 3a 63 6f
0050 64 65 0a 4e 52 55 72 6d 45 65 6e 66 71 55 42 44
0060 51 57 53 55 4a 45 31 57 4e 57 7a 64 6b 59 52 45
0070 68 6c 57 67 58 70 4d 45 4d 6b 46 45 2b 67 46 45
0080 32 69 4f 46 61 6b 48 68 5a 66 30 77 58 72 75 68
0090 57 63 38 47 38 54 54 51 2b 33 6b 6a 78 43 55 57
00a0 5a 49 4b 73 35 6e 70 6b 65 6e 4f 77 73 68 43 5a
00b0 31 6c 6b 6d 66 4e 4e 39 45 33 6d 47 39 71 39 35
00c0 72 30 70 56 64 53 53 38 69 71 30 6b 6c 6e 46 58
00d0 61 69 75 30 54 5a 70 39 36 6d 50 53 72 76 6b 52
00e0 31 55 31 72 6c 36 72 2f 68 34 78 65 73 31 70 78
00f0 76 6e 47 6c 51 5a 6a 51 4b 49 37 36 79 66 53 2b
0100 4a 56 6c 2b 47 74 74 41 58 65 42 2f 65 41 51 5a
0110 6e 53 47 6f 59 44 79 64 38 7a 67 72 78 34 49 70
0120 45 45 45 6b 73 71 4b 55 61 65 4c 56 43 2f 58 70
0130 43 6d 33 50 78 78 6f 31 54 2b 35 56 31 46 5a 71


Frame 35: 85 bytes on wire (680 bits), 85 bytes captured (680 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: 10000, Dst Port: 46288, Seq: 106438, Ack: 116, Len: 19
Data (19 bytes)
Data: 00116f7065726174696f6e3a636c6f73650a0a
[Length: 19]


0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00
0010 00 47 35 53 40 00 40 06 07 5c 7f 00 00 01 7f 00
0020 00 01 27 10 b4 d0 88 6d 12 d6 e9 15 1e 27 80 18
0030 02 00 fe 3b 00 00 01 01 08 0a 3f d3 fe ff 3f d3
0040 fe fe 00 11 6f 70 65 72 61 74 69 6f 6e 3a 63 6c
0050 6f 73 65 0a 0a



Mejoras adicionales implementadas:

En este apartado enumeramos las mejoras implementadas y una breve explicación de cómo se han hecho.

1. Explorar ficheros remotos mediante nickname (browse <nick>)
   Para implementar esta mejora modificamos el comando browse <IP:port> y el comando fgserve <port>. Primero, para que esto funcione el directorio tiene que tener una lista de los servidores que hay en escucha, con sus nicks y sus respectivas IP y puertos de escucha. Para eso, cada vez que un cliente de nanoFiles ejecuta el comando fgserve, y antes de empezar a servir, envía un mensaje al directorio indicandole que está escuchando clientes en el puerto <port> con el nick que tiene registrado en el directorio. El directorio comprueba que no haya nada igual ya registrado y lo almacena en un mapa de servidores y le manda una confirmación. Una vez hecho esto el cliente ya puede empezar a servir, y los clientes que quieran buscar ese servidor por el nick con browse <nick> mandarán un mensaje al directorio preguntando por ese nick. Si no lo encuentra, el directorio enviará un mensaje de error, y si lo hace enviará la IP y el puerto de ese servidor para que el cliente pueda conectarse.

2. Ampliar userlist con información sobre servidores
   Una vez realizado browse<nick> esta mejora es muy sencilla. Como el directorio guarda la información de los peers que hay conectados, cuando comprueba todos

los nicks que tiene guardados para enviar la userlist a un cliente, simplemente comprueba si los nicks están guardados en el mapa de servidores y para el que lo esté, añade “ - S” después del nick, para indicar que es un servidor.

3. Mantener actualizados ficheros y servidores disponibles
   No implementamos filelist así que no podemos mantener actualizada la lista de ficheros, pero sí la de servidores. Para esto, cuando un peer servidor ejecuta fgstop y detiene el servidor, envía a su vez un mensaje al directorio para comunicarle que ha dejado de servir y que este lo retire del mapa de servidores. Así la próxima vez que alguien solicite la lista de usuarios, no saldrá al lado del nick del peer que estaba sirviendo la cadena “ - S”.

4. Consultar ficheros disponibles en el explorador (queryfiles)
   Para esta mejora nos decidimos por un formato similar al que usa el directorio para userlist. En vez de mandar los datos “raw”, decidimos que era mejor mandar una cadena ya formateada con toda la información preparada para mostrarlo directamente por pantalla. Para queryfiles, el cliente le envía una solicitud queryfiles al servidor, y este consulta su base de datos (NanoFiles.db) para obtener todos los ficheros que está sirviendo y manda de cada fichero su filehash, su nombre y su tamaño en bytes. Cuando le llega el paquete al cliente este solo tiene que imprimirlo por pantalla. Decidimos que solo mande un paquete con datos al igual que userlist porque, si bien esto no es del todo correcto porque si tiene demasiados ficheros un servidor, o un directorio tiene muchos usuarios, la capacidad máxima de 64K es bastante alta y en el contexto de las prácticas de esta asignatura no es algo que se vaya a dar, sin embargo nos suponía un esfuerzo de diseño extra y no teníamos suficiente tiempo, especialmente para implementar esto en userlist, ya que UDP no es confiable y sería difícil hacer confiable la entrega de varios paquetes en serie.

5. Detener servidor en primer plano (fgstop)
   Para realizar esta mejora simplemente le ponemos un timeout arbitrario al serverSocket que recibe solicitudes de conexión al servidor (8 segundos), y cada vez que salta el timeout un buffer de lectura comprueba si se ha escrito algo por entrada. Si lee fgstop, entonces detiene el servidor.

Conclusiones

Como conclusión, estamos satisfechos con la implementación de la práctica porque hemos conseguido implementar la funcionalidad básica correctamente así como unas cuantas mejoras, sin embargo creemos que deberíamos haber dedicado un poco más de tiempo a la misma para implementar el servidor concurrente en segundo plano que es la mejora significativa, pero no nos ha sido posible. Aún así el desarrollo de la práctica nos ha ayudado a comprender cómo funciona una comunicación UDP/TCP entre procesos y

también a mejorar nuestro proceso de diseño de software ya que es una aplicación relativamente compleja que tiene bastantes partes y todas deben funcionar como se espera.