REDES DE COMUNICACIONES

Proyecto NanoFiles. Documento de diseño.

Ángel Sáez Rosique
angel.saezr@um.es
Francisco Javier Ramírez López
fj.ramirezlopez@um.es

Universidad de Murcia
Facultad de Informática
Grupo 2

Curso Académico 2023/2024

ÍNDICE

1. Introducción.

NanoFiles es un sistema de compartición y transferencia de archivos utilizando el lenguaje de programación Java. Este sistema está constituido por un servidor de directorio (denominado Directory) y un conjunto de peers o pares (programa NanoFiles) que interactúan tanto mediante un modelo cliente-servidor como un modelo P2P. En este sistema, los pares pueden actuar tanto como clientes del servidor de directorio para registrar usuarios y consultar información sobre archivos disponibles, como servidores de archivos para permitir a otros peers descargar dichos archivos. Además, como hemos implementado la mejora de poder lanzar el servidor de ficheros en segundo plano, NanoFiles permite que un peer pueda actuar simultáneamente como cliente y servidor.

A lo largo del desarrollo de NanoFiles, hemos implementado la autenticación de usuarios, el intercambio de claves de sesión, la publicación y consulta de archivos compartidos, y la descarga de archivos desde otros peers, entre otras cosas. También explicamos en este documento cómo hemos implementado una serie de mejoras a la hora de diseñar el protocolo y al implementar el programa.

Por otro lado, en este documento se especifica el diseño de los protocolos peer-to-peer y de comunicación con el directorio. El diseño de estos protocolos facilita la autenticación de usuarios, la publicación de archivos, y permite que cada peer actúe de manera dinámica como cliente y servidor según las necesidades del sistema. Asimismo, este documento detalla cómo cada tipo de mensaje se emplea para gestionar interacciones específicas entre peers y entre peers y el servidor de directorio.

2. Formato de los mensajes del protocolo de comunicación con el directorio.

Para definir el protocolo de comunicación con el Directorio, hemos utilizado mensajes textuales con formato “campo:valor”. El valor que tome el campo “operation” indicará el tipo de mensaje y por tanto su formato (qué campos vienen a continuación).

2.1. Tipos y descripción de los mensajes.

Mensaje	Sentido de la comunicación
login	Cliente → Directorio

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
nickname:angel\n
\n

Mensaje	Sentido de la comunicación
login_ok	Directorio → Cliente

Mensaje enviado por el directorio de NanoFiles al cliente para confirmar que el inicio de sesión se ha realizado exitosamente. Se devuelve la sessionKey asociada a dicho cliente. Ejemplo:

operation:login_ok\n
sessionkey:4781\n
\n

Mensaje	Sentido de la comunicación
login_failed	Directorio → Cliente

Mensaje que envía el directorio de NanoFiles al cliente para informar de que no se ha podido iniciar sesión correctamente. Ejemplo:

operation:login_failed\n
\n

Mensaje	Sentido de la comunicación
userlist	Cliente → Directorio

Lo envía el cliente de NanoFiles para que obtener del directorio la lista de nombres de usuario (nicknames) que hay registrados en el servidor de directorio. Ejemplo:

operation:userlist\n
\n

Mensaje	Sentido de la comunicación
userlist_response	Directorio → Cliente

Este mensaje es enviado por el directorio de NanoFiles para mostrar la lista de nicknames que hay registrados en el servidor de directorio. Como hemos implementado la mejora de ampliar el comando `userlist` para que indique qué usuarios registrados están actuando como servidores, este mensaje lo indica con la etiqueta [server].

Ejemplo:

operation:userlist_response\n
users:angel,francisco[server],alicia\n

\n

Mensaje	Sentido de la comunicación
register_server	Cliente → Directorio

Mensaje que envía el cliente de NanoFiles al directorio para registrar en el directorio un servidor de ficheros, si no estaba ya registrado. También se usa para dar de baja el servidor de ficheros del directorio, ya que si el cliente que envía este mensaje tenía registrado su servidor de ficheros, el directorio da de baja la información asociada a dicho servidor. Ejemplo:

operation:register_server\n
sessionkey:4781\n
port:10000\n
\n

Mensaje	Sentido de la comunicación
register_server_ok	Directorio → Cliente

Este mensaje lo envía el directorio de NanoFiles al cliente para confirmar que se ha registrado el servidor de ficheros o que se ha eliminado correctamente toda la información asociada al servidor de ficheros. Ejemplo:

operation:register_server_ok\n
\n

Mensaje	Sentido de la comunicación
register_server_failed	Directorio → Cliente

Este mensaje lo envía el directorio de NanoFiles al cliente para informar de que no se ha podido registrar el servidor de ficheros o que no se ha podido dar de baja. Ejemplo:

operation:register_server_failed\n
\n

Mensaje	Sentido de la comunicación
logout	Cliente → Directorio

Lo envía el cliente de NanoFiles al directorio para cerrar sesión en el directorio, dando de baja el nombre de usuario registrado en el directorio anteriormente. Ejemplo:

operation:logout\n
nickname:angel\n
sessionkey:4781\n
\n

Mensaje	Sentido de la comunicación
logout_ok	Directorio → Cliente

Mensaje enviado por el directorio de NanoFiles al cliente para confirmar que se ha cerrado la sesión del cliente en el directorio. Ejemplo:

operation:logout_ok\n
\n

Mensaje	Sentido de la comunicación
logout_failed	Directorio → Cliente

Lo envía el directorio de NanoFiles al cliente para informar de que no se ha podido cerrar la sesión del cliente en el directorio. Ejemplo:

operation:logout_failed\n
\n

Mensaje	Sentido de la comunicación
lookup_username	Cliente → Directorio

Lo envía el cliente de NanoFiles al directorio para obtener del directorio la dirección de socket asociada a un determinado nickname. Ejemplo:

operation:lookup_username\n
nickname:angel\n
\n

Mensaje	Sentido de la comunicación
lookup_response	Directorio → Cliente

Este mensaje lo envía el directorio de NanoFiles al cliente para devolver la dirección de socket (IP,puerto) asociada a un determinado nickname. Ejemplo:

operation:lookup_response\n
addr:127.0.0.1,10000\n
\n

Mensaje	Sentido de la comunicación
publish_files	Cliente → Directorio

Mensaje que envía el cliente de NanoFiles al directorio para publicar en el directorio los metadatos de los ficheros que este peer tiene en su carpeta compartida. Ejemplo:

operation:publish_files\n
sessionkey:4781\n
file:hash1,name1,size1\n
file:hash2,name2,size2\n
…
\n

Mensaje	Sentido de la comunicación
publish_files_ok	Directorio → Cliente

Este mensaje lo envía el directorio de NanoFiles al cliente para confirmar que se han publicado en el directorio los metadatos de los ficheros que este peer tiene en su carpeta compartida. Ejemplo:

operation:publish_files_ok\n
\n

Mensaje	Sentido de la comunicación
publish_files_failed	Directorio → Cliente

Mensaje enviado por el directorio de NanoFiles al cliente para informar de que no se ha podido publicar en el directorio los metadatos de los ficheros que este peer tiene en su carpeta compartida. Ejemplo:

operation:publish_files_failed\n
\n

Mensaje	Sentido de la comunicación
filelist	Cliente → Directorio

Lo envía el cliente de NanoFiles para que el servidor muestre la lista de ficheros que hay publicados en el servidor de directorio. Ejemplo:

operation:filelist\n
\n

Mensaje	Sentido de la comunicación
filelist_response	Directorio → Cliente

Este mensaje es enviado por el servidor de NanoFiles para mostrar la lista de ficheros que hay publicados en el servidor de directorio. Ejemplo:

operation:filelist_response\n
file:hash1,name1,size1\n
file:hash2,name2,size2\n
…
\n

Mensaje	Sentido de la comunicación
search	Cliente → Directorio

Lo envía el cliente de NanoFiles para obtener los nicknames de los servidores que tienen disponible un determinado fichero identificado por su hash. Ejemplo:

operation:search\n
hash:hash1\n
\n

Mensaje	Sentido de la comunicación
search_response	Directorio → Cliente

Este mensaje es enviado por el servidor de NanoFiles para mostrar la lista de nicknames que tienen disponible un determinado fichero identificado por su hash. Ejemplo:

operation:search_response\n
users:angel,francisco,alicia\n

\n

3. Formato de los mensajes del protocolo peer-to-peer.

Para definir el protocolo de comunicación entre peers, 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.

3.1. Tipo y descripción de mensajes.

Mensaje	Sentido de la comunicación
FileNotFound	Servidor de ficheros → Cliente

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	Sentido de la comunicación
Download	Cliente → Servidor de ficheros

Mensaje enviado por el par cliente al par servidor de ficheros para solicitar la descarga de un fichero del servidor. Ejemplo:

Opcode (1 byte)	Longitud del campo (4 bytes)	Campo (32 bytes)	Longitud del campo (4 bytes)	Campo (32 bytes)
2	32	Hash del fichero a descargar	32	Nombre del fichero a descargar

Mensaje	Sentido de la comunicación
File	Servidor de ficheros → Cliente

Lo envía el par servidor de ficheros al par cliente para devolver un fragmento del archivo que se solicitó descargar. Ejemplo:

Opcode (1 byte)	Longitud del campo (4 bytes)	Campo (32 bytes)
3	32	Datos del fichero a descargar

Mensaje	Sentido de la comunicación
EndOfFile	Servidor de ficheros → Cliente

Este mensaje lo envía el par servidor de ficheros al par cliente para indicar que ha terminado de enviar el archivo que se solicitó descargar. Ejemplo:

Opcode (1 byte)	Longitud del campo (4 bytes)	Campo (32 bytes)
4	32	Hash del fichero enviado

4. Autómatas de protocolo.

Respecto a los autómatas, hemos considerado las siguientes restricciones:

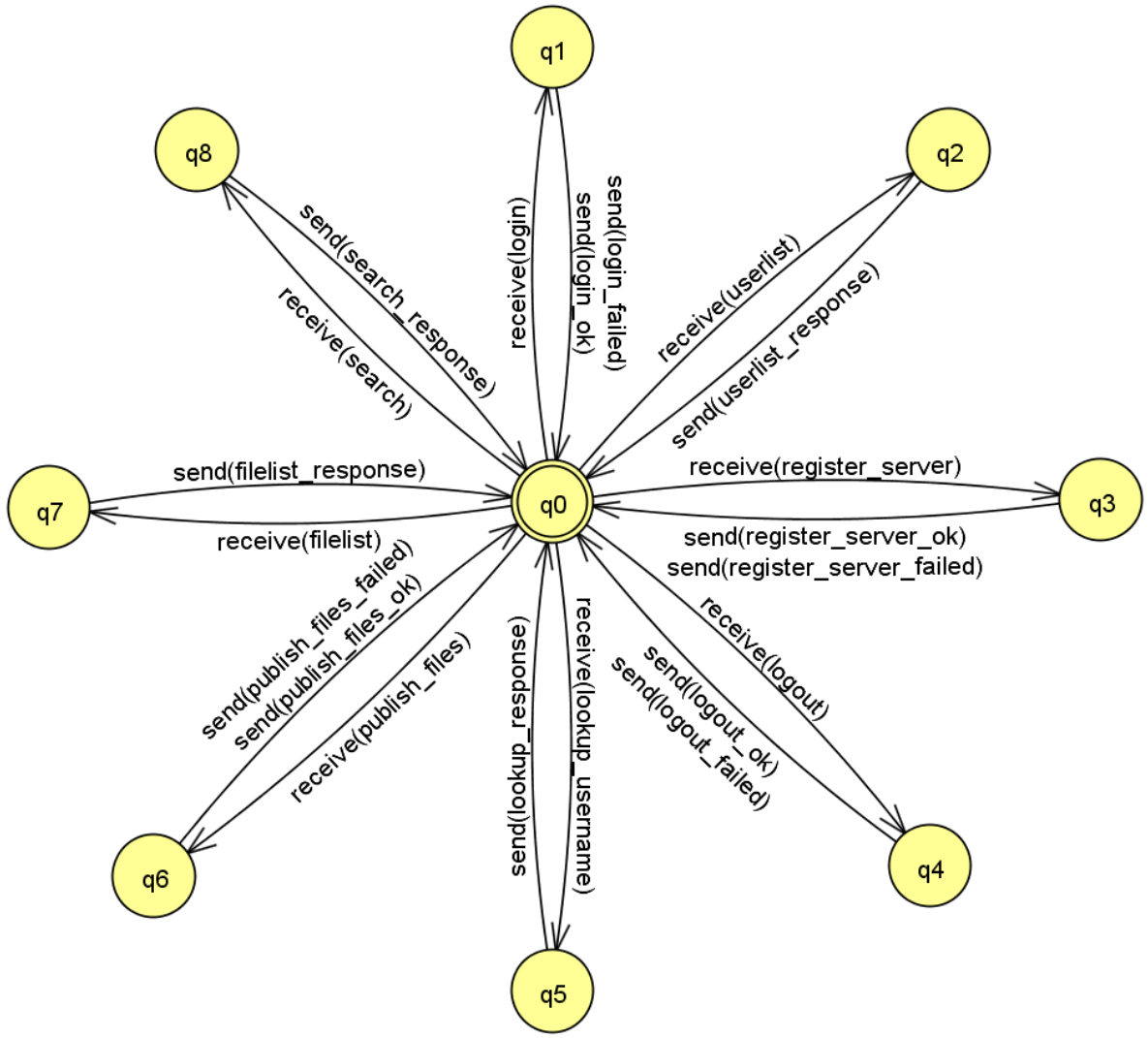
- Un cliente no puede iniciar sesión si ya está logueado en el directorio.
- Un cliente no puede ejecutar logout si no está logueado en el directorio. Tampoco puede ejecutarlo si el servidor en segundo plano está ejecutándose, por lo que tendrá que parar el servidor primero.
- Un cliente no puede solicitar la lista de usuarios al directorio si no ha iniciado sesión, ya que no tendrá la dirección del directorio. Lo mismo pasa con los mensajes “filelist”, “lookup_username” y “search”.
- Tampoco podrá descargar ficheros si no ha iniciado sesión, ya que al implementar la mejora de download por nick, el cliente podría intentar solicitar la dirección de un servidor al directorio y, al no haber iniciado sesión, no tendría la dirección del directorio.
- Un cliente no puede registrar el servidor en el directorio sin haber iniciado sesión, ya que se necesita la sessionKey.
- Tampoco podrá registrar el servidor si ya lo ha hecho, ya que hemos implementado el comando stopserver, que envía un mensaje “register_server” al directorio para dar

de baja el servidor asociado a dicha sessionKey. Por lo tanto, al volver a enviar “register_server” para el mismo servidor, el directorio elimina el servidor de su lista de servidores.

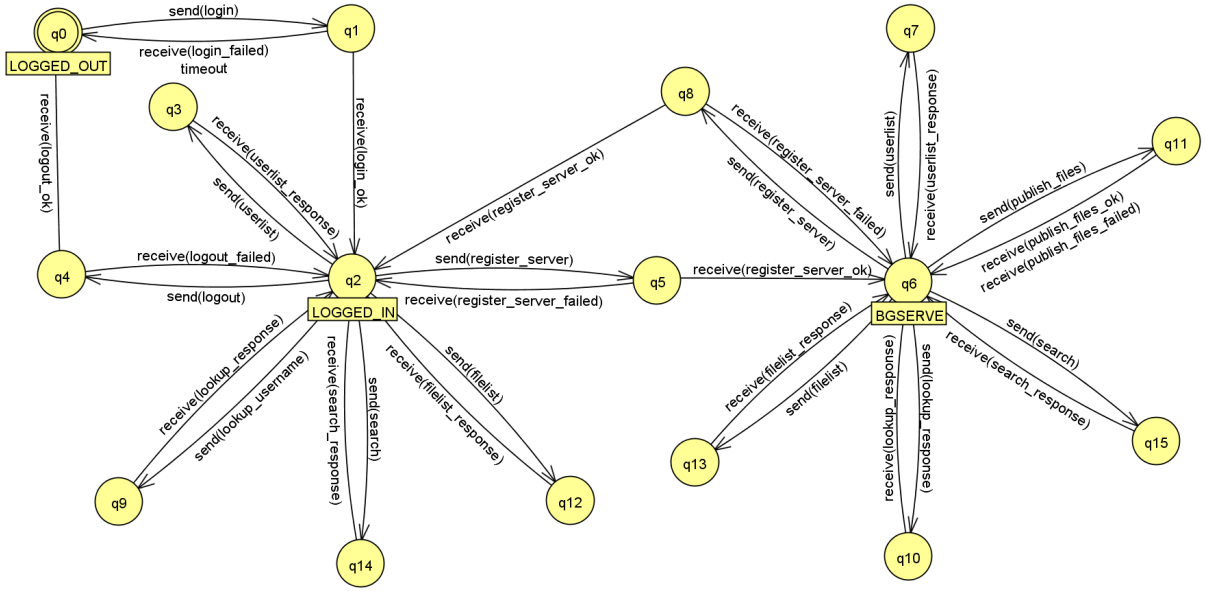
- Un cliente no puede publicar los ficheros que está compartiendo si no ha iniciado sesión o si no ha registrado en el directorio el servidor en segundo plano.

4.1. Autómata para el servidor de directorio.

Este autómata muestra cómo actúa el directorio al recibir peticiones de los peers.



4.2. Autómata para un peer cuando actúa como cliente del directorio.



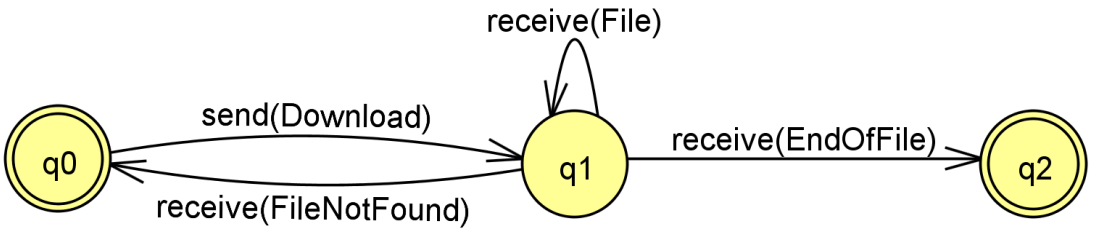
Se puede observar que cuando un peer actúa como cliente del directorio, el cliente puede estar en tres estados distintos:

- **LOGGED_OUT:** en este estado los peers únicamente pueden iniciar sesión en el directorio. Es el estado inicial de los peers al ejecutar el programa Nanofiles, aunque también se puede llegar a este estado al cerrar sesión.

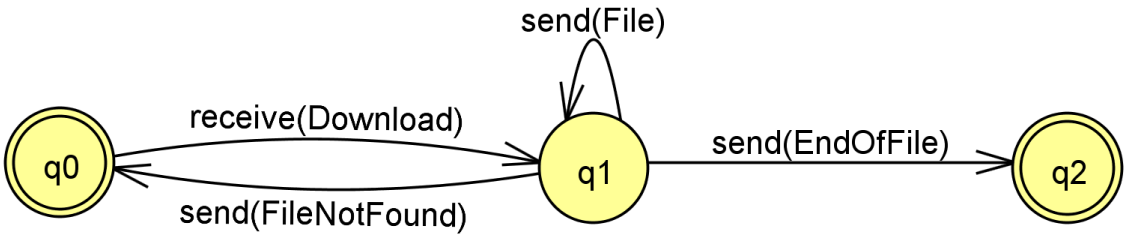
- **LOGGED_IN:** se llega a este estado al iniciar sesión o al parar el servidor de ficheros en segundo plano. En este estado se pueden ejecutar la mayoría de comandos, excepto aquellos que requieren no haber iniciado sesión (login) y los que requieren que el peer esté actuando como servidor (publish). Al ejecutar el comando bgserve se envía el mensaje register_server y si el directorio informa de que se ha registrado correctamente el servidor, se pasa al estado BGSERVE.

- **BGSERVE:** este estado indica que el peer está actuando como servidor de ficheros. Sin embargo, como el servidor de ficheros está corriendo en segundo plano, el peer puede seguir actuando a la vez como cliente y ejecutar la mayoría de los comandos que se pueden ejecutar en el estado LOGGED_IN, excepto logout, ya que se necesita parar el servidor primero.

4.3. Autómata para un peer cuando actúa como cliente de otro peer servidor.



4.4. Autómata para un peer cuando actúa como servidor de otro peer cliente.



En los dos anteriores autómatas podemos observar que el procedimiento para descargar cualquier fichero es que un peer cliente envía una solicitud a un peer servidor a través del mensaje Download. Una vez que el peer servidor procesa la solicitud, puede responder con el mensaje FileNotFound, indicando que no se ha encontrado el archivo o que ha ocurrido algún otro error. Pero si la solicitud es correcta, comienza a enviar fragmentos del fichero al peer cliente mediante el mensaje File y una vez ha enviado completamente el archivo, envía el mensaje EndOfFile para informar de ello.

5. Mejoras implementadas.

Las mejoras que hemos implementado en el proyecto de NanoFiles son las siguientes:

5.1. `fgserve` puerto variable - 0,5 ptos.

En esta mejora se modifica el comando `fgserve` para que, si el puerto de escucha predeterminado (10000) no está disponible, el servidor pueda utilizar otros números de puerto disponibles (10001, 10002, etc.). De esta manera, será posible ejecutar múltiples instancias del programa NanoFiles en la misma máquina funcionando como servidores.

5.2. `stopserver` - 0,5 ptos.

La función de esta mejora es detener el servidor en segundo plano que anteriormente se lanzó con el comando `bgserve`. Además, también se informa al directorio mediante el mensaje “register_server” para que elimine el servidor de su lista de servidores. El directorio únicamente elimina servidores al recibir el mensaje “register_server” si la sessionKey del mensaje se corresponde con la de algún servidor registrado.

5.3. `bgserve` puerto efímero - 0,5 ptos.

Hemos modificado el comando `bgserve` para que el servidor pueda utilizar cualquier número de puerto disponible para escuchar peticiones, en lugar de un puerto específico predefinido.

5.4. `userlist` ampliado con servidores - 0,5 ptos.

La opción básica del comando `userlist` es mostrar la lista de nombres de usuario (nicknames) que hay registrados en el servidor de directorio. A partir de esta mejora hemos ampliado el comando `userlist` para que muestre cuáles de los usuarios en la lista son servidores. Para implementar esta mejora, hemos tenido que implementar que el directorio compruebe para cada usuario registrado si su sessionKey se encuentra en la lista de servidores registrados.

5.5. `publish` + `filelist` - 0,5 ptos.

La mejora `publish` se encarga de informar al directorio sobre los archivos compartidos por un peer. Esta mejora, por sí sola, no tiene utilidad, ya que no existía previamente ningún comando que usase esta información almacenada en el directorio. Por ello, hemos implementado también el comando `filelist`, que muestra la lista de todos los archivos que los peers han publicado en el servidor de directorio. Para cada archivo, se indica su nombre, tamaño y hash.

5.6. `publish` + `search` - 0,5 ptos.

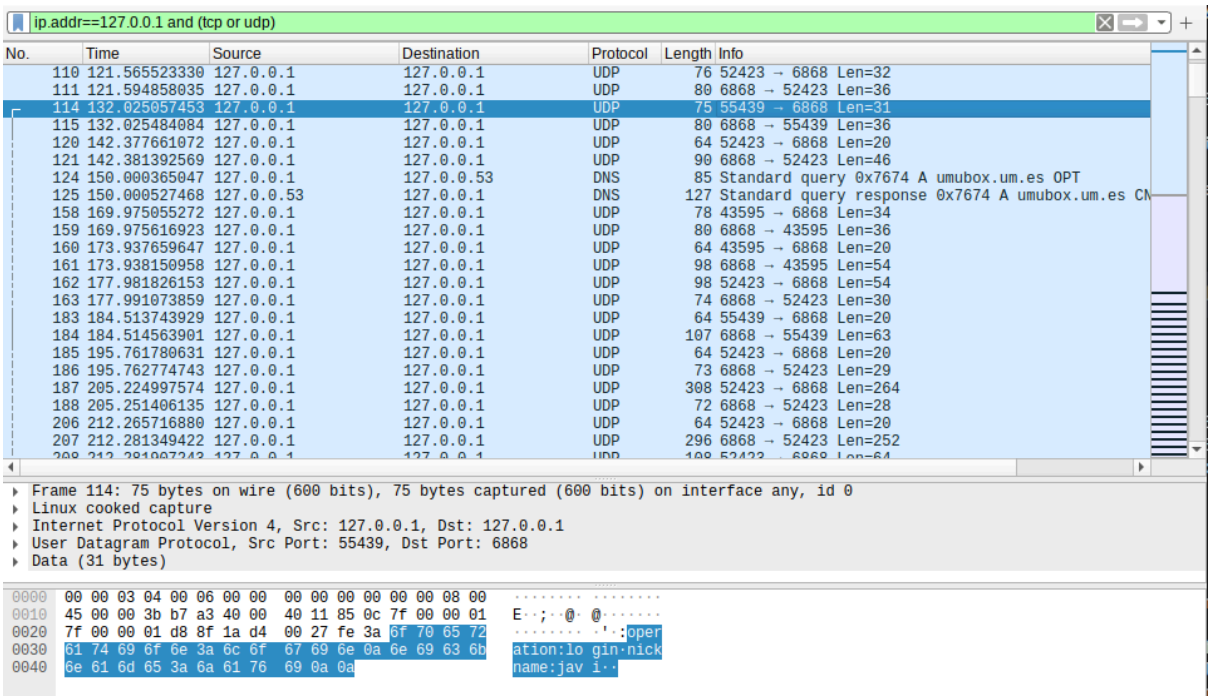
Al igual que con `filelist`, la mejora `publish` también va unida a la mejora `search`. Esta mejora nos muestra el nickname de los servidores que tienen disponible un determinado fichero identificado por su hash.

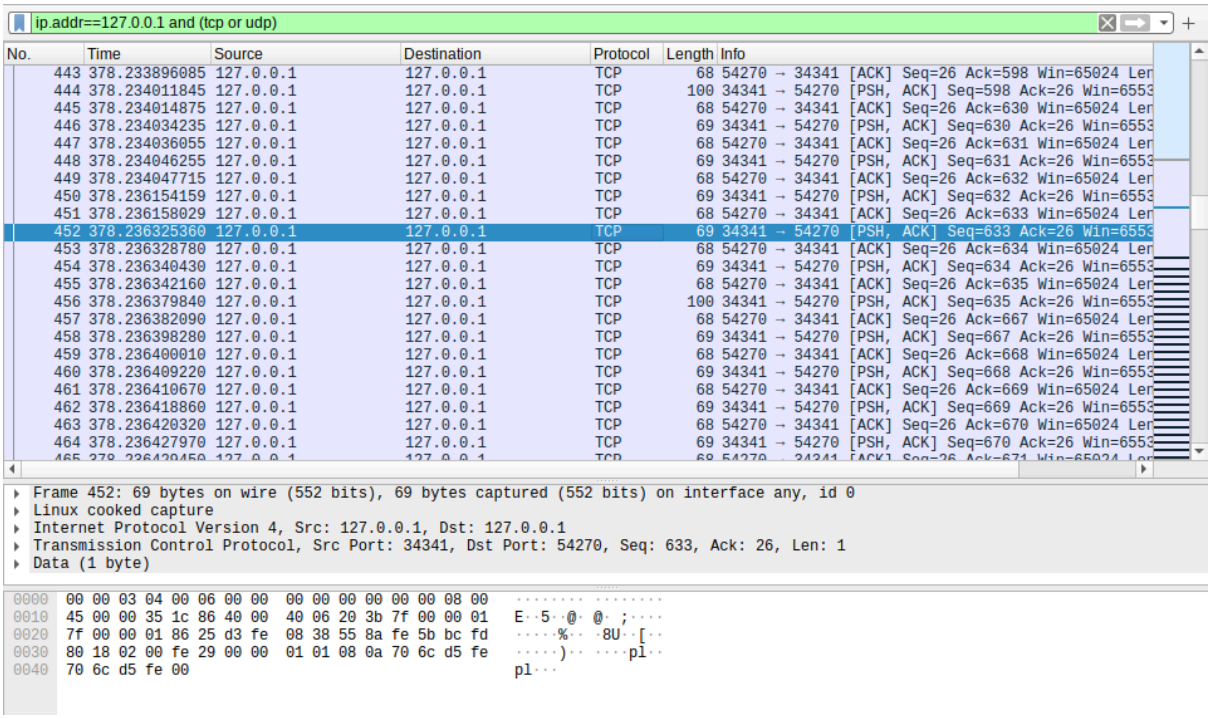
5.7. `filelist` ampliado con servidores - 0,5 ptos.

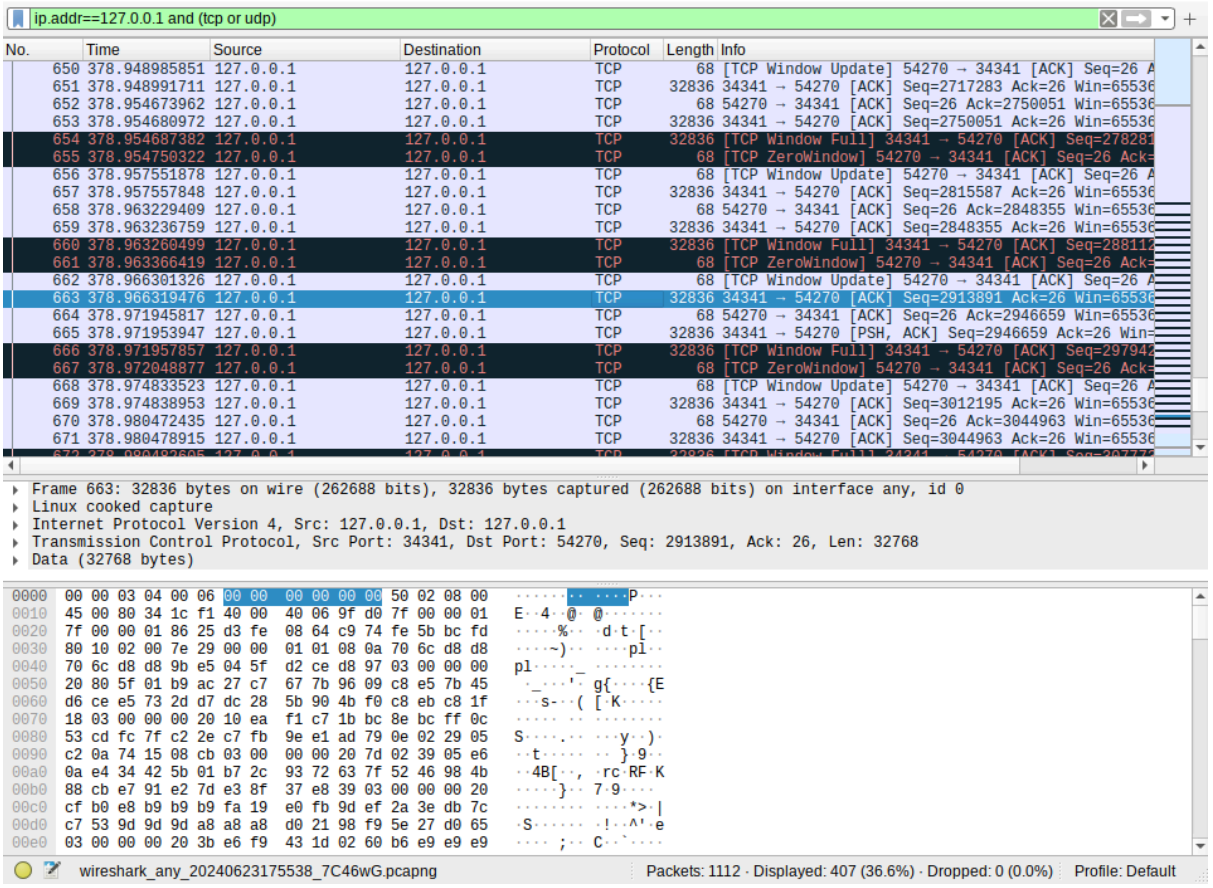
Hemos ampliado el comando `filelist` para mostrar, junto a cada archivo disponible, la lista de direcciones de los servidores que lo están compartiendo, además del nombre, tamaño y hash del archivo. Para ello, el cliente primero obtiene la lista de ficheros y para cada uno comprueba qué servidor lo está compartiendo con el mensaje “search”. Además, como necesita mostrar la dirección del nickname que devuelve el directorio en el mensaje “search_response”, envía un mensaje “lookup_username” para cada nickname para obtener su dirección.

6. Ejemplo de intercambio de mensajes.

A continuación mostraremos el intercambio de mensajes mediante Wireshark:




ip.addr==127.0.0.1 and (tcp or udp)
No. Time Source Destination Protocol Length Info
443 378.233896085 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=598 Win=65024 Len
444 378.234011845 127.0.0.1 127.0.0.1 TCP 100 34341 → 54270 [PSH, ACK] Seq=598 Ack=26 Win=6553
445 378.234014875 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=630 Win=65024 Len
446 378.234034235 127.0.0.1 127.0.0.1 TCP 69 34341 → 54270 [PSH, ACK] Seq=630 Ack=26 Win=6553
447 378.234036055 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=631 Win=65024 Len
448 378.234046255 127.0.0.1 127.0.0.1 TCP 69 34341 → 54270 [PSH, ACK] Seq=631 Ack=26 Win=6553
449 378.234047715 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=632 Win=65024 Len
450 378.236154159 127.0.0.1 127.0.0.1 TCP 69 34341 → 54270 [PSH, ACK] Seq=632 Ack=26 Win=6553
451 378.236158029 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=633 Win=65024 Len
452 378.236325360 127.0.0.1 127.0.0.1 TCP 69 34341 → 54270 [PSH, ACK] Seq=633 Ack=26 Win=6553
453 378.236328780 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=634 Win=65024 Len
454 378.236340430 127.0.0.1 127.0.0.1 TCP 69 34341 → 54270 [PSH, ACK] Seq=634 Ack=26 Win=6553
455 378.236342160 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=635 Win=65024 Len
456 378.236379840 127.0.0.1 127.0.0.1 TCP 100 34341 → 54270 [PSH, ACK] Seq=635 Ack=26 Win=6553
457 378.236382090 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=667 Win=65024 Len
458 378.236398280 127.0.0.1 127.0.0.1 TCP 69 34341 → 54270 [PSH, ACK] Seq=667 Ack=26 Win=6553
459 378.236400010 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=668 Win=65024 Len
460 378.236409220 127.0.0.1 127.0.0.1 TCP 69 34341 → 54270 [PSH, ACK] Seq=668 Ack=26 Win=6553
461 378.236410670 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=669 Win=65024 Len
462 378.236418860 127.0.0.1 127.0.0.1 TCP 69 34341 → 54270 [PSH, ACK] Seq=669 Ack=26 Win=6553
463 378.236420320 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=670 Win=65024 Len
464 378.236427970 127.0.0.1 127.0.0.1 TCP 69 34341 → 54270 [PSH, ACK] Seq=670 Ack=26 Win=6553
Frame 452: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface any, id 0
Linux cooked capture
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 34341, Dst Port: 54270, Seq: 633, Ack: 26, Len: 1
Data (1 byte)
0000 00 00 03 04 00 06 00 00 00 00 00 00 00 00 08 00
0010 45 00 00 35 1c 86 40 00 40 06 20 3b 7f 00 00 01
0020 7f 00 00 01 86 25 d3 fe 08 38 55 8a fe 5b bc fd
0030 80 18 02 00 fe 29 00 00 01 01 08 0a 70 6c d5 fe
0040 70 6c d5 fe 00


ip.addr==127.0.0.1 and (tcp or udp)
No. Time Source Destination Protocol Length Info
650 378.948985851 127.0.0.1 127.0.0.1 TCP 68 [TCP Window Update] 54270 → 34341 [ACK] Seq=26 A
651 378.948991711 127.0.0.1 127.0.0.1 TCP 32836 34341 → 54270 [ACK] Seq=2717283 Ack=26 Win=65536
652 378.954673962 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=2750051 Win=65536
653 378.954680972 127.0.0.1 127.0.0.1 TCP 32836 34341 → 54270 [ACK] Seq=2750051 Ack=26 Win=65536
654 378.954687382 127.0.0.1 127.0.0.1 TCP 32836 [TCP Window Full] 34341 → 54270 [ACK] Seq=27828
655 378.954750322 127.0.0.1 127.0.0.1 TCP 68 [TCP ZeroWindow] 54270 → 34341 [ACK] Seq=26 Ack=
656 378.957551878 127.0.0.1 127.0.0.1 TCP 68 [TCP Window Update] 54270 → 34341 [ACK] Seq=26 A
657 378.957557848 127.0.0.1 127.0.0.1 TCP 32836 34341 → 54270 [ACK] Seq=2815587 Ack=26 Win=65536
658 378.963229409 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=2848355 Win=65536
659 378.963236759 127.0.0.1 127.0.0.1 TCP 32836 34341 → 54270 [ACK] Seq=2848355 Ack=26 Win=65536
660 378.963260499 127.0.0.1 127.0.0.1 TCP 32836 [TCP Window Full] 34341 → 54270 [ACK] Seq=28811
661 378.963366419 127.0.0.1 127.0.0.1 TCP 68 [TCP ZeroWindow] 54270 → 34341 [ACK] Seq=26 Ack=
662 378.966301326 127.0.0.1 127.0.0.1 TCP 68 [TCP Window Update] 54270 → 34341 [ACK] Seq=26 A
663 378.966319476 127.0.0.1 127.0.0.1 TCP 32836 34341 → 54270 [ACK] Seq=2913891 Ack=26 Win=65536
664 378.971945817 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=2946659 Win=65536
665 378.971953947 127.0.0.1 127.0.0.1 TCP 32836 34341 → 54270 [PSH, ACK] Seq=2946659 Ack=26 Win=
666 378.971957857 127.0.0.1 127.0.0.1 TCP 32836 [TCP Window Full] 34341 → 54270 [ACK] Seq=29794
667 378.972048877 127.0.0.1 127.0.0.1 TCP 68 [TCP ZeroWindow] 54270 → 34341 [ACK] Seq=26 Ack=
668 378.974833523 127.0.0.1 127.0.0.1 TCP 68 [TCP Window Update] 54270 → 34341 [ACK] Seq=26 A
669 378.974838953 127.0.0.1 127.0.0.1 TCP 32836 34341 → 54270 [ACK] Seq=3012195 Ack=26 Win=65536
670 378.980472435 127.0.0.1 127.0.0.1 TCP 68 54270 → 34341 [ACK] Seq=26 Ack=3044963 Win=65536
671 378.980478915 127.0.0.1 127.0.0.1 TCP 32836 34341 → 54270 [ACK] Seq=3044963 Ack=26 Win=65536
Frame 663: 32836 bytes on wire (262688 bits), 32836 bytes captured (262688 bits) on interface any, id 0
Linux cooked capture
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 34341, Dst Port: 54270, Seq: 2913891, Ack: 26, Len: 32768
Data (32768 bytes)
0000 00 00 03 04 00 06 00 00 00 00 00 00 50 02 08 00
0010 45 00 80 34 1c f1 40 00 40 06 9f d0 7f 00 00 01
0020 7f 00 00 01 86 25 d3 fe 08 64 c9 74 fe 5b bc fd
0030 80 10 02 00 7e 29 00 00 01 01 08 0a 70 6c d8 d8
0040 70 6c d8 d8 9b e5 04 5f d2 ce d8 97 03 00 00 00
0050 20 80 5f 01 b9 ac 27 c7 67 7b 96 09 c8 e5 7b 45
0060 d6 ce e5 73 2d d7 dc 28 5b 90 4b f0 c8 eb c8 1f
0070 18 03 00 00 00 20 10 ea f1 c7 1b bc 8e bc ff 0c
0080 53 cd fc 7f c2 2e c7 fb 9e e1 ad 79 0e 02 29 05
0090 c2 0a 74 15 08 cb 03 00 00 00 20 7d 02 39 05 e6
00a0 0a e4 34 42 5b 01 b7 2c 93 72 63 7f 52 46 98 4b
00b0 88 cb e7 91 e2 7d e3 8f 37 e8 39 03 00 00 00 20
00c0 cf b0 e8 b9 b9 b9 fa 19 e0 fb 9d ef 2a 3e db 7c
00d0 c7 53 9d 9d 9d a8 a8 a8 d0 21 98 f9 5e 27 d0 65
00e0 03 00 00 00 20 3b e6 f9 43 1d 02 60 b6 e9 e9 e9
wireshark_any_20240623175538_7C46wG.pcapng
Packets: 1112 · Displayed: 407 (36.6%) · Dropped: 0 (0.0%)
Profile: Default

7. (Opcional) Enlace a grabación de pantalla mostrando los programas en funcionamiento.

Se proporciona el siguiente enlace a la grabación de pantalla mostrando el funcionamiento de NanoFiles, además se ha compartido el vídeo al profesor mediante OneDrive de la Universidad de Murcia.

nanofiles.mp4

8. Conclusiones.

Implementar NanoFiles ha implicado desarrollar funcionalidades esenciales como la autenticación de usuarios, el intercambio de claves de sesión, la publicación y consulta de archivos compartidos, y la descarga de archivos desde otros peers.

Por lo tanto, el desarrollo de esta práctica nos ha permitido aplicar y ampliar nuestros conocimientos sobre diseño de protocolos y programación en Java, así como entender mejor los conceptos teóricos de la asignatura. Sobre todo, consideramos que hemos afianzado los conocimientos teóricos que teníamos sobre los protocolos TCP y UDP.