

ESTRUCTURA PROYECTO REDES 2024/25



PABLO MEROÑO HERRANZ
HUGO EGEA JARA

Grupo:2.4
Profesor: JOSÉ RUBÉN TITOS GIL

ÍNDICE:

1. Introducción
2. Formato de los mensajes del protocolo de comunicación con el directorio.
3. Formatos de los mensajes del protocolo de transferencia de archivos.
4. Autómatas de protocolos.
5. Mejoras implementadas y breve descripción sobre su programación
6. Capturas de Whireshark
7. Conclusiones

1. Introducción.

En este documento se especifica el diseño de dos protocolos de comunicación de nivel de aplicación del sistema NanoFiles: uno para la comunicación entre cada peer y el directorio, que será un protocolo confiable semi-duplex basado en parada y espera, que operará sobre un protocolo de nivel de transporte no confiable como UDP; y otro para la comunicación entre dos peers, sabiendo que el protocolo de nivel de transporte es confiable (TCP).

Las mejoras que hemos considerado a la hora de diseñar el protocolo son las de upload, serve con puerto efímero, filelist ampliado con servidores y quit que actualiza ficheros y servidores.

2. 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).

Tipos y descripción de los mensajes

Mensaje: **PING**

Sentido de la comunicación: Cliente→ Directorio

Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para comprobar que el servidor de directorio está activo y usa un protocolo compatible con el del peer

```
operation:ping\n
protocol:<protocolID>\n
\n
```

Mensaje: **SUCCESSFUL**

Sentido de la comunicación: Directorio→ Cliente

Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles como mensaje de confirmación para comunicar que el ping ha sido exitoso y que el cliente se ha registrado como servidor y se han subido sus ficheros cuando ejecuta serve. También para indicar que un servidor de ficheros se ha dado de baja con éxito.

```
operation:successful\n
\n
```

Mensaje: **ERROR**

Sentido de la comunicación: Directorio→ Cliente

Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles como mensaje de error para comunicar que el ping ha fallado, que un cliente no puede descargar ficheros porque no ha subido antes los suyos, y para comunicar que no se ha podido registrar al cliente como servidor y por tanto no se han subido sus ficheros.

```
operation:Error\n
\n
```

Mensaje: **FILELIST**

Sentido de la comunicación: Cliente→ Directorio

Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para obtener la lista de ficheros que los peers servidores han publicado al directorio.

```
operation:filelist\n\n
```

Mensaje: **AVAILABLE_FILES**

Sentido de la comunicación: Directorio→ Cliente

Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles como respuesta al mensaje FILELIST. Envía la lista de ficheros que los peers servidores han publicado al directorio con información de cada fichero como su nombre, tamaño, hash y los peers servidores que lo tienen.

```
operation:availableFiles\nfilename:<filename> \nsize:<file size>\nhash:<file hash> \nservers:<list of servers>\n...\n
```

Mensaje: **SERVE**

Sentido de la comunicación: Cliente→ Directorio

Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para publicar los ficheros que el peer sirve al resto. Para cada fichero se indica el nombre, tamaño y hash.

Ejemplo:

```
operation:serve\nport:<port number>\nfilename:<filename>\nsize:<file size>\nhash:<file_hash> \n...\n
```

Mensaje: **DOWNLOAD**

Sentido de la comunicación: Cliente→ Directorio

Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para indicar que quiere descargar un fichero cuyo nombre contiene la subcadena indicada en el campo filename.

```
operation:download\n
filename:<subcadena>\n
\n
```

Mensaje: **SERVERS_SHARING_THIS_FILE**

Sentido de la comunicación: Directorio→ Cliente

Descripción: Este mensaje lo envía el Directorio al cliente de NanoFiles como respuesta al mensaje DOWNLOAD. Envía la lista de ficheros que los peers servidores han publicado al directorio cuyo nombre contiene la subcadena indicada en el mensaje download. Para cada fichero se indica su nombre y el servidor que lo tiene.

```
operation:serversSharingThisFile\n
filename:<filename>\n
server:<server>\n
...
\n
```

Mensaje: **UNREGISTER**

Sentido de la comunicación: Cliente→ Directorio

Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para indicar que quiere darse de baja como servidor de ficheros en el puerto indicado.

```
operation:unregister\n
port:<port number>\n
/n
```

3. 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: **DownLoadFile (opcode = 1)**

Sentido de la comunicación: Cliente → Servidor de ficheros

Descripción: Este mensaje lo envía el par cliente al par servidor de ficheros (receptor) para indicar una subcadena del nombre del fichero que quiere descargar.

Ejemplo:

Opcode (1 byte)	Longitud (1 byte)	Valor (n bytes)
0x01	0x18	ejerciciosDeProgramación

Mensaje: **FileFound (opcode = 2)**

Sentido de la comunicación: Servidor de ficheros → Cliente

Descripción: Este mensaje lo envía el par servidor de ficheros al par cliente (receptor) para indicar que ha encontrado el fichero del mensaje de petición de descarga.

Ejemplo:

Opcode (1 byte)	Hash (SHA-1) (20 bytes)	FileSize (Long: 8 byte)
0x02	3c2727cbe32f267231fa1b2d04aa905a0e4c21d9	0x00 0x00 0x00 0x00 0x00 0x20 0x01 0x00

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

Sentido de la comunicación: Servidor de ficheros → Cliente

Descripción: Este mensaje lo envía el par servidor de ficheros al par cliente (receptor) para indicar que no ha encontrado ningún fichero cuyo nombre contenga la subcadena del mensaje de petición de descarga.

Ejemplo:

Opcode (1 byte)
0x03

Mensaje: **AmbiguousName (opcode = 4)**

Sentido de la comunicación: Servidor de ficheros → Cliente

Descripción: Este mensaje lo envía el par servidor de ficheros al par cliente (receptor) para indicar que la subcadena del mensaje de petición de descarga es ambiguo pues coincide con el nombre de varios.

Ejemplo:

Opcode (1 byte)
0x04

Mensaje: GetChunk (opcode = 5)

Sentido de la comunicación: Cliente → Servidor de ficheros

Descripción: Este mensaje lo envía el par cliente al par servidor de ficheros (receptor) para indicar desde qué posición y que tamaño de porción quiere descargar de un fichero previamente acordado.

Ejemplo:

Opcode (1 byte)	File offset (Long: 8 byte)	Chunk size (Int: 4 bytes)
0x05	0x00 0x00 0x00 0x00 0x00 0x20 0x01 0x00	0x01 0x00 0x00 0x00

Mensaje: GiveChunk (opcode = 6)

Sentido de la comunicación: Servidor de ficheros → Cliente y Cliente → Servidor de ficheros

Descripción: Este mensaje lo envía el par servidor de ficheros al par cliente (receptor) para enviarle trozos con el tamaño indicado del fichero acordado en el mensaje de descarga. También lo puede enviar el cliente al servidor para subir los trozos de un fichero acordado en el mensaje de subida.

Ejemplo:

Opcode (1 byte)	Longitud (Int: 4 bytes)	Valor (n bytes)
0x06	0x00 0x10 0x00 0x00	...

Mensaje: UploadFile (opcode = 7)

Sentido de la comunicación: Cliente → Servidor de ficheros

Descripción: Este mensaje lo envía el par cliente al par servidor de ficheros (receptor) para indicar una subcadena del nombre del fichero que quiere subir.

Ejemplo:

Opcode (1 byte)	Longitud (1 byte)	Valor (n bytes)
0x07	0x18	ejerciciosDeProgramación

Mensaje: CloseConnection (opcode = 8)

Sentido de la comunicación: Servidor de ficheros → Cliente

Descripción: Este mensaje lo envía el par cliente al par servidor de ficheros (receptor) cuando ya ha recibido todos los chunks necesarios y así cortar la conexión. También se usa cuando un par cliente está subiendo un fichero a otro cliente y ya ha terminado de enviar todos los chunks.

Ejemplo:

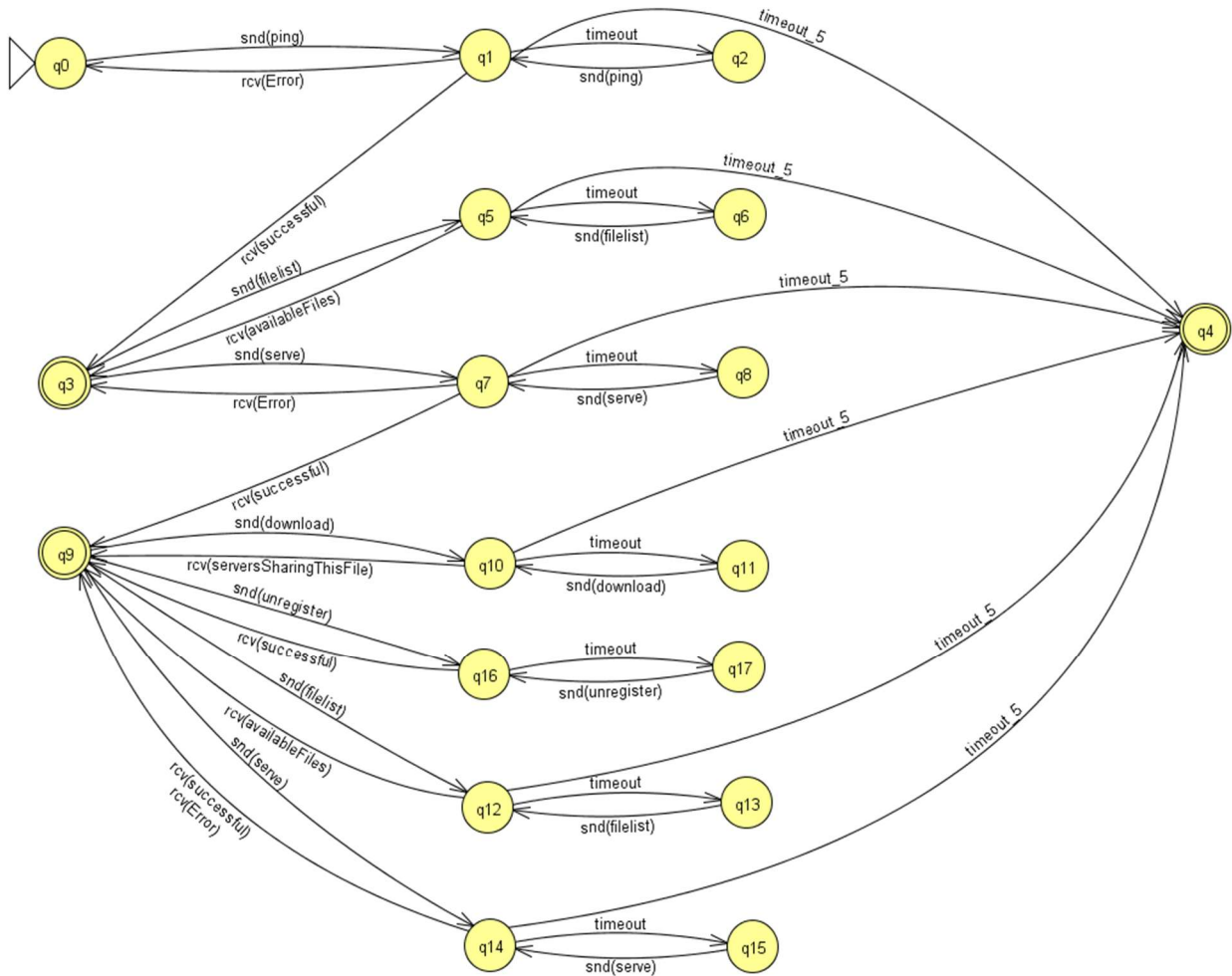
Opcode (1 byte)
0x08

4. Autómatas de protocolo

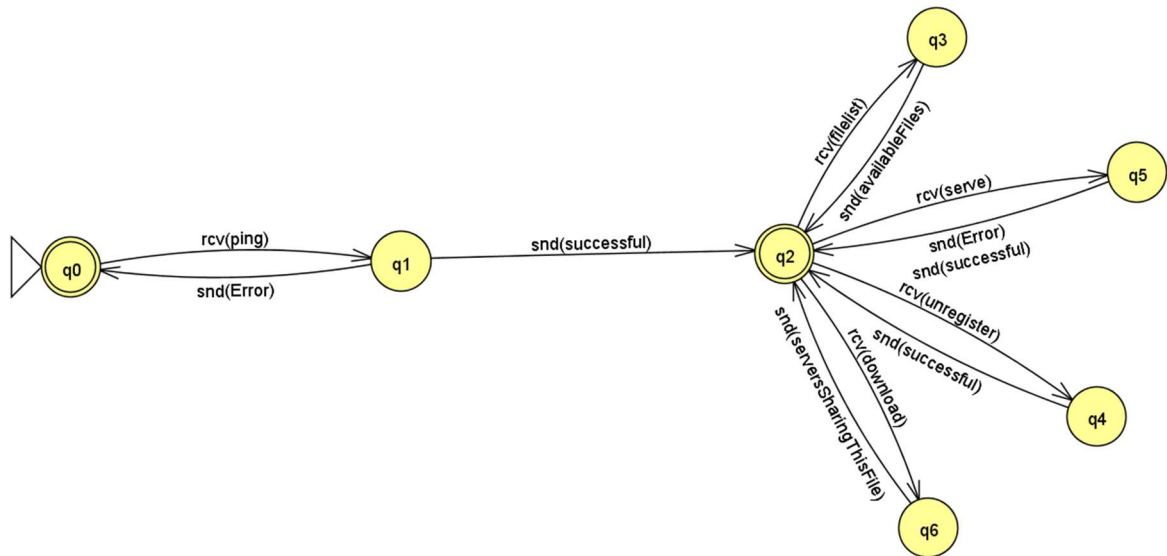
Con respecto a los autómatas, hemos considerado las siguientes restricciones:

- Un cliente del directorio no puede realizar ninguna operacion relacionada con los ficheros si antes no ha hecho un ping exitoso con el Directorio.
- Un cliente de ficheros no puede descargar ningún fichero proporcionado por otro peer servidor si antes no ha publicado sus ficheros.
- Un cliente de ficheros no puede subir ningún fichero a otro cliente si este último no está sirviendo ya sus ficheros.
- Para cualquier mensaje que el cliente envía al Directorio, si se producen 5 timeouts se llega a un estado final.

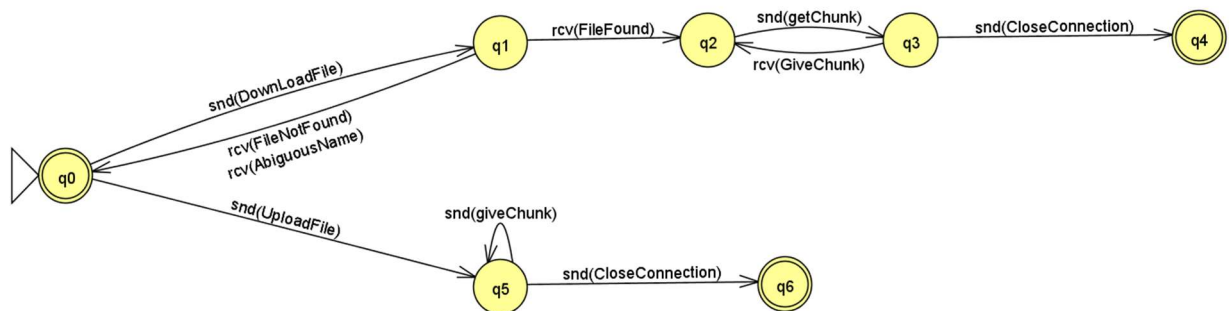
Autómata rol cliente de directorio



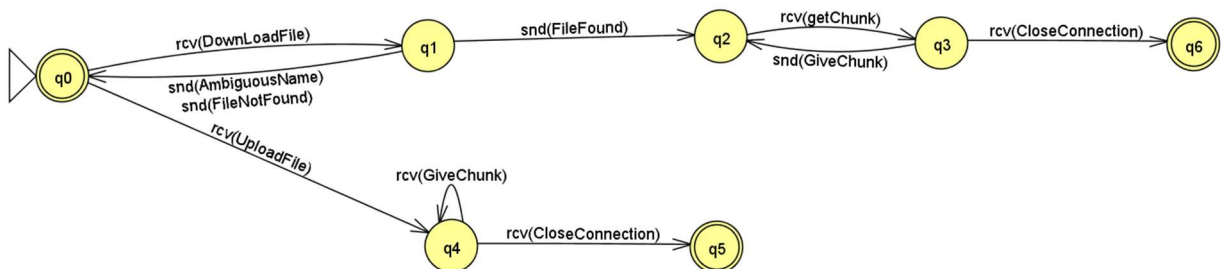
Autómata rol servidor de directorio



Autómata rol cliente de ficheros



Autómata rol servidor de ficheros



5. Mejoras implementadas y breve descripción sobre su programación

Mejora: upload

Descripción:

Se ha implementado la funcionalidad que permite a un peer enviar (subir) archivos a otro peer que actúa como servidor. Esta mejora amplía el protocolo TCP del sistema NanoFilesP2P para incluir la subida de ficheros.

Funcionamiento:

- **Lado cliente (emisor):**
 - A través del comando upload, el cliente localiza el fichero a subir desde su base de datos local (NanoFiles.db) utilizando una subcadena del nombre.
 - Si se encuentra un único fichero coincidente, se establece una conexión TCP con el servidor destino y se envía un mensaje OPCODE_UPLOAD_FILE indicando el nombre del fichero.
 - A continuación, se transmite el contenido del fichero en bloques binarios (chunks) mediante mensajes OPCODE_GIVE_CHUNK, y finalmente se cierra la conexión con OPCODE_CLOSE_CONNECTION.
- **Lado servidor (receptor):**
 - Al recibir el mensaje OPCODE_UPLOAD_FILE, el servidor crea un fichero local en su carpeta compartida y abre un FileOutputStream.
 - A medida que recibe los mensajes OPCODE_GIVE_CHUNK, escribe los datos en el fichero.
 - Al cerrarse la conexión, el servidor calcula el hash, tamaño y ruta del fichero recibido, y lo registra en su base de datos (NanoFiles.db).

Clases modificadas:

- NFController: interpreta el comando upload y lanza el proceso.
- NFControllerLogicP2P: implementa la lógica de subida en uploadFileToServer.
- NFServer: extiende el método serveFilesToClient para manejar la recepción de archivos y su integración en la base de datos.
- FileDataBase: se ha añadido el método public void addFile(FileInfo fileInfo) para añadir el nuevo fichero a la base de datos.

Mejora: Serve con puerto efímero

Descripción:

Se ha modificado el sistema para permitir que un peer sirva archivos usando un puerto efímero TCP, es decir, uno asignado dinámicamente por el sistema operativo.

Funcionamiento:

- **Inicio del servidor:**
 - En la clase NFControllerLogicP2P, el método startFileServer() instancia un objeto NFServer pasando el puerto 0.
 - Dentro del constructor de NFServer, se usa serverSocket.bind(new InetSocketAddress(port)). Si el puerto es 0, el sistema escoge uno libre automáticamente.
 - Luego, se actualiza this.port con serverSocket.getLocalPort() para registrar el puerto realmente asignado.

- **Comunicación con el directorio:**

- Tras iniciar el servidor, el método `registerFileServer` en `NFControllerLogicDir` obtiene el puerto real mediante `controllerPeer.getServerPort()` y lo incluye en el mensaje `DirMessage` al directorio.
- El mensaje servido enviado al directorio incluye ese puerto dinámico, lo que permite a otros peers conectarse correctamente.

Clases modificadas:

- `NFControllerLogicP2P` -> método `startFileServer()`: inicia `NFServer` con puerto efímero (`port = 0`).
- `NFServer` -> constructor actualiza `this.port` tras el `bind`.
- `NFControllerLogicDir` -> usa el puerto dinámico para registrar al peer servidor.

Mejora: filelist ampliado con servidores

Descripción:

Se ha extendido el funcionamiento del comando `filelist` para que no solo devuelva la lista de ficheros disponibles en el sistema, sino también las direcciones IP y puertos de los servidores (peers) que los comparten. Esto permite a los usuarios saber de antemano quién tiene qué archivo antes de descargarlo.

Funcionamiento:

- **Respuesta del directorio:**

- En `NFDIrectoryServer` se utiliza una estructura para registrar los archivos que comparte cada peer:
`public static Map<InetSocketAddress, ArrayList<FileInfo>> addrMapFiles.`
- Cuando un cliente ejecuta el comando `serve`, se le registra en este mapa con su dirección y la lista de ficheros que sirve.
- Al recibir una solicitud con `OPERATION_FILELIST`, se recorre este mapa para generar un nuevo mapa invertido:
`Map<FileInfo, List<InetSocketAddress>> availableFiles;`
- Este nuevo mapa asocia cada fichero con la lista de peers que lo tienen.
- Este mapa se envía dentro de un objeto `DirMessage` como respuesta al cliente.

- **Cliente: mostrar información ampliada:**

- El cliente, en el método `getAndPrintFileList()` de `NFControllerLogicDir`, recibe este mapa y lo imprime con un nuevo método añadido a la clase `FileInfo`:
`FileInfo.printToSysoutWithServers(trackedFiles);`
- Este método muestra en consola una tabla con columnas:
 - Nombre del fichero
 - Tamaño
 - Hash
 - Lista de servidores que lo tienen
- Se decidió añadir este método en vez de modificar el método `public static void printToSysout(FileInfo[] files)`, por el simple hecho de que este último se usa en otros casos y su modificación los complicaría.

Mejora: Quit que actualiza ficheros y servidores

Descripción:

Se ha implementado una mejora en el comando quit para que, al salir de la aplicación, el peer se dé de baja correctamente como servidor de ficheros en el directorio. Esto evita que otros peers intenten conectarse a un servidor inexistente, manteniendo la integridad del sistema.

Funcionamiento:

- **Al ejecutar el comando quit:**
 - En NFController, al procesar el comando COM_QUIT, se comprueba si el servidor TCP está activo (controllerPeer.serving()).
 - Si es así, se detiene el servidor (controllerPeer.stopFileServer()).
 - A continuación, se llama a controllerDir.unregisterFileServer() para comunicar al directorio que este peer ya no está disponible.
- **Desregistro en el directorio:**
 - En NFControllerLogicDir, el método unregisterFileServer() crea un mensaje DirMessage de tipo OPERATION_UNREGISTER, incluyendo el puerto del servidor (obtenido mediante controllerPeer.getServerPort()).
 - En el NFDirectoryServer, al recibir este mensaje, elimina del mapa addrMapFiles la entrada correspondiente a esa dirección.

6. Capturas de Whireshark

Envía Mensaje Ping

Wireshark interface showing a packet capture on interface eth0. The capture shows a sequence of packets, including a ping request (ICMP Echo) and its corresponding reply.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	155.54.224.55	155.54.224.54	UDP	78	59177 → 6868 Len=36
2	0.001441361	155.54.224.54	155.54.224.55	UDP	64	6868 → 59177 Len=22
3	0.000000000	Cisco_3b:b4:14	Spanning-tree-(for-...	STP	119	MST. Root = 16384/0/7c:69:f6:66:fd:80 Cost = 2000 Port = 0x8014
4	0.694313907	Cisco_3b:b4:14	Cisco_3b:b4:14	LOOP	60	Reply
5	2.371843190	Cisco_3b:b4:14	Spanning-tree-(for-...	STP	119	MST. Root = 16384/0/7c:69:f6:66:fd:80 Cost = 2000 Port = 0x8014

Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface eth0, id 60000
Ethernet II, Src: ASUSTekCOMPU_d5:0a:51 (2c:4d:54:d5:0a:51), Dst: ASUSTekCOMPU_d5:1b:58 (2c:4d:54:d5:1b:58)
Internet Protocol Version 4, Src: 155.54.224.55, Dst: 155.54.224.54
User Datagram Protocol, Src Port: 59177, Dst Port: 6868
Data (36 bytes)
Data: 6f706572617469666e3a70696e670a70726f746f636f6c3a313233343536373839410a0a
[Length: 36]

Recibe Successful

Wireshark interface showing a packet capture on interface eth0. The capture shows a sequence of packets, including a ping request (ICMP Echo) and its corresponding reply.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	155.54.224.55	155.54.224.54	UDP	78	59177 → 6868 Len=36
2	0.001441361	155.54.224.54	155.54.224.55	UDP	64	6868 → 59177 Len=22
3	0.000000000	Cisco_3b:b4:14	Spanning-tree-(for-...	STP	119	MST. Root = 16384/0/7c:69:f6:66:fd:80 Cost = 2000 Port = 0x8014
4	0.694313907	Cisco_3b:b4:14	Cisco_3b:b4:14	LOOP	60	Reply
5	2.371843190	Cisco_3b:b4:14	Spanning-tree-(for-...	STP	119	MST. Root = 16384/0/7c:69:f6:66:fd:80 Cost = 2000 Port = 0x8014

Frame 2: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface eth0, id 60001
Ethernet II, Src: ASUSTekCOMPU_d5:1b:58 (2c:4d:54:d5:1b:58), Dst: ASUSTekCOMPU_d5:0a:51 (2c:4d:54:d5:0a:51)
Internet Protocol Version 4, Src: 155.54.224.54, Dst: 155.54.224.55
User Datagram Protocol, Src Port: 6868, Dst Port: 59177
Data (22 bytes)
Data: 6f706572617469666e3a737563665737366756c0a0a
[Length: 22]

Envía Mensaje Filelist

*eth0

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Dell_7c:e0:5f	ASUSTekCOMPU_d5:0f:...	ARP	60	Who has 155.54.224.39? Tell 155.54.224.61
2	0.208611092	155.54.224.55	155.54.224.54	UDP	62	59177 → 6868 Len=20
3	0.210570422	155.54.224.54	155.54.224.55	UDP	68	6868 → 59177 Len=26
4	1.024012119	Dell_7c:e0:5f	ASUSTekCOMPU_d5:0f:...	ARP	60	Who has 155.54.224.39? Tell 155.54.224.61
5	1.071310691	Cisco_3b:b4:14	Spanning-tree-(for-...	STP	119	MST. Root = 16384/0/7c:69:f6:66:fd:80 Cost = 2000 Port = 0x8014
6	1.682415680	Cisco_3b:b4:14	Cisco_3b:b4:14	LOOP	60	Reply

Frame 2: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface eth0, id 60

Ethernet II, Src: ASUSTekCOMPU_d5:0a:51 (2c:4d:54:d5:0a:51), Dst: ASUSTekCOMPU_d5:1b:58 (2c:4d:54:d5:1b:58)

Internet Protocol Version 4, Src: 155.54.224.55, Dst: 155.54.224.54

User Datagram Protocol, Src Port: 59177, Dst Port: 6868

Data (20 bytes)

Data: 0f7085726174696f6e3a66696c656c6973740a0a [Length: 20]

0000 2c 4d 54 d5 1b 58 2c 4d 54 d5 0a 51 08 00 45 00 ,MT..X,M T..Q..E-
0010 00 30 e2 58 40 00 40 11 61 89 9b 36 e0 37 9b 36 .0.X@.@.a..6.7.6
0020 e0 36 e7 29 1a d4 00 1c 3c 25 6f 70 65 72 61 74 .6.)....<operat
0030 69 6f 6e 3a 66 69 6c 65 6c 69 73 74 0a 0a ion:file list..

Recibe AvailableFiles

*eth0

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Dell_7c:e0:5f	ASUSTekCOMPU_d5:0f:...	ARP	60	Who has 155.54.224.39? Tell 155.54.224.61
2	0.208611092	155.54.224.55	155.54.224.54	UDP	62	59177 → 6868 Len=20
3	0.210570422	155.54.224.54	155.54.224.55	UDP	68	6868 → 59177 Len=26
4	1.024012119	Dell_7c:e0:5f	ASUSTekCOMPU_d5:0f:...	ARP	60	Who has 155.54.224.39? Tell 155.54.224.61
5	1.071310691	Cisco_3b:b4:14	Spanning-tree-(for-...	STP	119	MST. Root = 16384/0/7c:69:f6:66:fd:80 Cost = 2000 Port = 0x8014
6	1.682415680	Cisco_3b:b4:14	Cisco_3b:b4:14	LOOP	60	Reply

Frame 3: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface eth0, id 60

Ethernet II, Src: ASUSTekCOMPU_d5:1b:58 (2c:4d:54:d5:1b:58), Dst: ASUSTekCOMPU_d5:0a:51 (2c:4d:54:d5:0a:51)

Internet Protocol Version 4, Src: 155.54.224.54, Dst: 155.54.224.55

User Datagram Protocol, Src Port: 6868, Dst Port: 59177

Data (26 bytes)

Data: 0f7085726174696f6e3a617061696c61626c6546696c65738a0a [Length: 26]

0000 2c 4d 54 d5 0a 51 2c 4d 54 d5 1b 58 08 00 45 00 ,MT..Q,M T..X..E-
0010 00 36 1c 88 40 00 40 11 27 54 9b 36 e0 36 9b 36 .6.@@.'T.6.6.6
0020 e0 37 1a d4 e7 29 00 22 f7 0e 6f 70 65 72 61 74 .7...)'.operat
0030 69 6f 6e 3a 61 70 61 69 6c 61 62 6c 65 46 69 6c ion:availablFil
0040 65 73 0a 0a es.

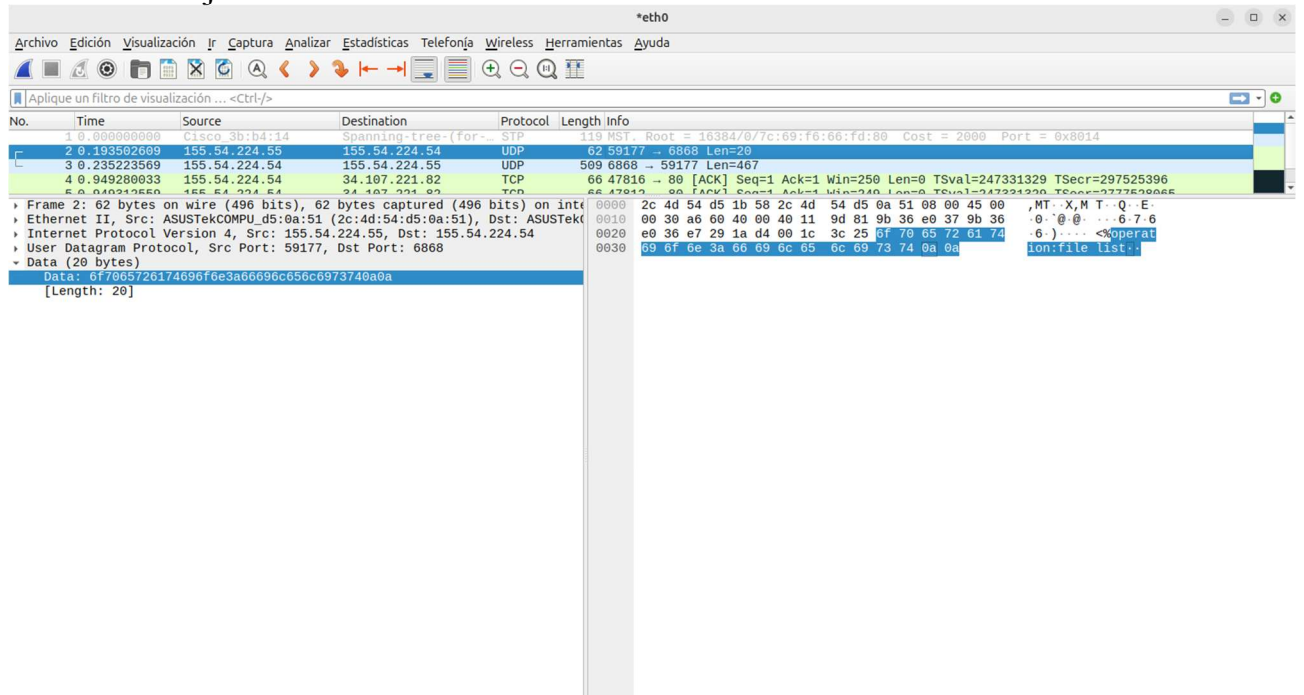
Envía Mensaje Serve

The screenshot shows a Wireshark packet capture on the interface *eth0. The packet list on the left shows five packets. Packet 1 is a UDP packet from 155.54.224.53 to 155.54.224.54, length 357. Packet 2 is a UDP packet from 155.54.224.54 to 155.54.224.53, length 22. Packet 3 is a spanning-tree protocol packet. Packet 4 is a MDNS query for PTR _ftp._tcp.local. Packet 5 is a MDNS query for PTR _nfs._tcp.local. The packet details pane shows the structure of the first packet: Ethernet II, Internet Protocol Version 4, User Datagram Protocol, and Data (357 bytes). The data field is truncated, showing the first 357 bytes of the message. The packet bytes pane shows the raw data in hexadecimal and ASCII. The ASCII column shows the message content: "MT-X,M T....E...@-B..6-5-6...f...m..7operat...ion:serve-port:3...993-fil ename:ar...hivo.2. docx:si...e:70792- hash:499...9422e0e5 ffd6d1ba...1b20e2d6 6f48f71f...5710c-fi lename:a...rchivo.3 .sql-siz...e:1556-h ash:3c27...27cbe32f 267231fa...1b2084aa 905a0e4c...2109-fil ename:ar...hivo.1. pdf:si...92300-h ash:288e...f9ca952e a0af157b...ce75b32e 4f55dbde...5013-fil ename:up...loaded- _archivo...4-size: 6845833...hash:587 e8d5011a...9b6e6923 24363132...45eb0ae6 e9030-".

Recibe Successful

The screenshot shows a Wireshark packet capture on the interface *eth0. The packet list on the left shows five packets. Packet 1 is a UDP packet from 155.54.224.53 to 155.54.224.54, length 357. Packet 2 is a UDP packet from 155.54.224.54 to 155.54.224.53, length 22. Packet 3 is a spanning-tree protocol packet. Packet 4 is a MDNS query for PTR _ftp._tcp.local. Packet 5 is a MDNS query for PTR _nfs._tcp.local. The packet details pane shows the structure of the first packet: Ethernet II, Internet Protocol Version 4, User Datagram Protocol, and Data (22 bytes). The data field is truncated, showing the first 22 bytes of the message. The packet bytes pane shows the raw data in hexadecimal and ASCII. The ASCII column shows the message content: "MT...M T..X.E...2..@-..M.6.6.6...5...f...operat...ion:succ essful..".

Envía Mensaje Filelist de nuevo



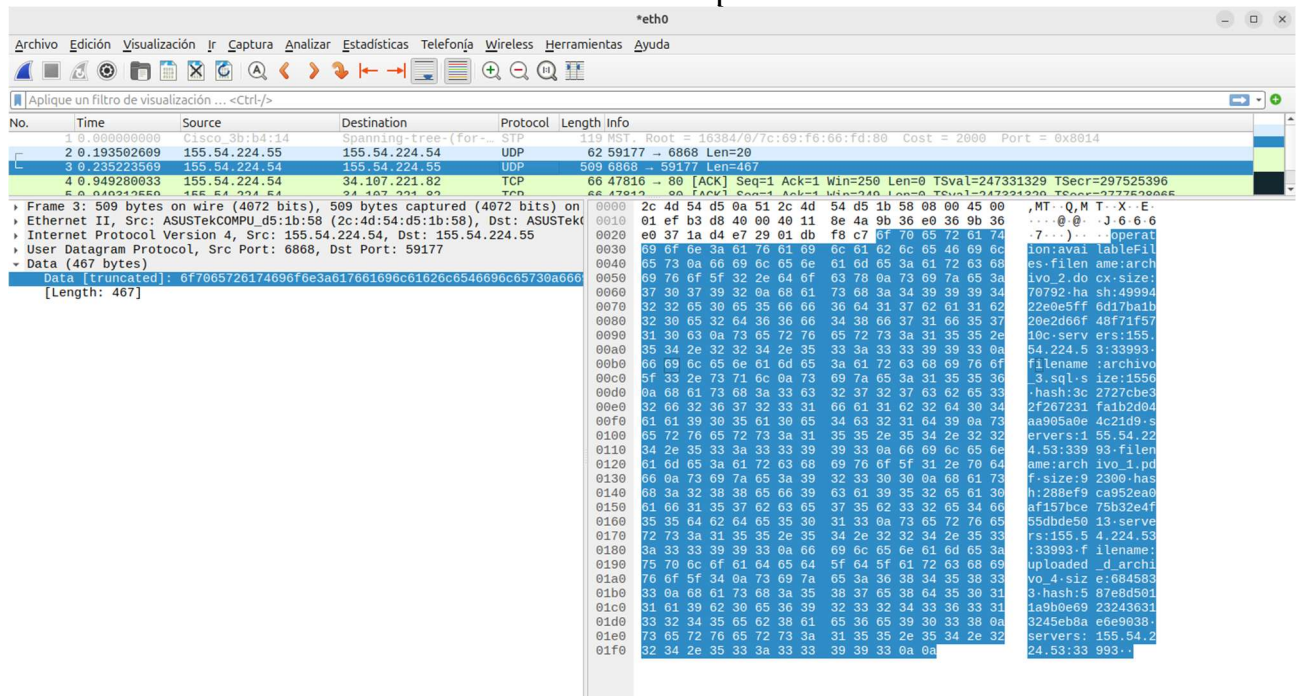
The image shows a Wireshark packet capture window titled '*eth0'. The packet list on the left shows a UDP packet (No. 2) from 155.54.224.55 to 155.54.224.54. The packet details on the right show the following structure:

- Frame 2: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface
- Ethernet II, Src: ASUSTekCOMPU_d5:0a:51 (2c:4d:54:d5:0a:51), Dst: ASUSTek
- Internet Protocol Version 4, Src: 155.54.224.55, Dst: 155.54.224.54
- User Datagram Protocol, Src Port: 59177, Dst Port: 6868
- Data (20 bytes)

The data field contains a file list message:

```
0000 2c 4d 54 d5 1b 58 2c 4d 54 d5 0a 51 08 00 45 00 ,MT..X.M.T..Q..E-  
0010 08 30 a6 60 40 00 40 11 9d 81 9b 36 e0 37 9b 36 ,0..@..6.7.6  
0020 e0 36 e7 29 1a d4 00 1c 3c 25 f7 70 05 72 01 7d ,6)....<Operat  
0030 69 6f 6e 3a 66 69 6c 65 6c 69 73 74 0a 0a ,on:file list..
```

Recibe AvailableFiles con los nuevos ficheros que está sirviendo



The image shows a Wireshark packet capture window titled '*eth0'. The packet list on the left shows a UDP packet (No. 2) from 155.54.224.55 to 155.54.224.54. The packet details on the right show the following structure:

- Frame 3: 509 bytes on wire (4072 bits), 509 bytes captured (4072 bits) on interface
- Ethernet II, Src: ASUSTekCOMPU_d5:1b:58 (2c:4d:54:d5:1b:58), Dst: ASUSTek
- Internet Protocol Version 4, Src: 155.54.224.55, Dst: 155.54.224.54
- User Datagram Protocol, Src Port: 6868, Dst Port: 59177
- Data (467 bytes)

The data field contains a file list message:

```
0000 2c 4d 54 d5 0a 51 2c 4d 54 d5 1b 58 00 00 45 00 ,MT..Q.M.T..X..E  
0010 01 ef b3 d8 40 00 40 11 8e 4a 9b 36 e0 36 9b 36 ,7...@...<operat  
0020 e0 37 1a d4 e7 29 01 db f8 c7 6f 70 65 72 61 7d ,...@...<operat  
0030 69 6f 6e 3a 61 76 61 69 6c 61 62 6c 65 40 69 6c ,on:aval lableFil  
0040 65 73 0a 66 69 6c 65 6e 61 6d 65 3a 61 72 63 68 ,es:file name:arch  
0050 69 76 6f 5f 32 2e 64 6f 63 78 0a 73 69 7a 65 3a ,ivo_2.do cx-size:  
0060 37 30 37 39 32 0a 68 61 73 68 3a 3a 39 39 39 34 ,70792-ha sh:49994  
0070 32 32 65 30 65 35 66 66 36 64 31 37 62 61 31 62 ,22e0e5ff 6d17ba1b  
0080 32 30 65 32 64 36 36 66 34 38 66 37 31 66 35 37 ,20e2d66f 48f71f57  
0090 31 30 63 0a 73 65 72 76 65 72 73 3a 31 35 35 2e ,10e-serv ers:155  
00a0 35 34 2e 32 32 34 2e 35 33 3a 33 39 39 39 0a ,54.224.5 3:33993  
00b0 69 69 6c 65 6e 61 6d 65 3a 61 72 63 68 69 76 6f ,filename :archivo  
00c0 5f 33 2e 73 71 6c 0a 73 69 7a 65 3a 31 35 35 36 ,.3.sql-s ize:1556  
00d0 0a 68 61 73 68 3a 33 63 32 37 32 37 63 62 65 33 ,.hash:3c 2727cbe3  
00e0 32 66 32 36 37 32 33 31 66 61 31 62 32 64 30 34 ,2f267231 fa1b2d04  
00f0 61 61 39 30 35 61 30 65 34 63 32 31 64 39 0a 73 ,aa905a0e 4c21d9-s  
0100 65 72 76 65 72 73 3a 31 35 35 2e 35 34 2e 32 32 ,ervers:1 55.54.22  
0110 34 2e 35 33 3a 33 33 39 39 33 0a 66 69 6c 65 6e ,4.53:339 93.file  
0120 01 6d 65 3a 61 72 63 68 69 76 6f 5f 31 2e 70 6d ,ams:arch ivo 1.pd  
0130 66 0a 73 69 7a 65 3a 39 32 33 30 0a 68 61 73 ,f-size:9 2300-has  
0140 68 3a 32 38 38 65 66 39 63 61 39 35 32 65 61 36 ,h:288ef9 ca952ea0  
0150 61 66 31 35 37 62 63 65 37 35 62 33 32 65 34 66 ,af157bce 75b32e4f  
0160 35 35 64 62 64 65 35 30 31 33 0a 73 65 72 76 65 ,55dbde50 13-serve  
0170 72 73 3a 31 35 35 2e 35 34 2e 32 32 34 2e 35 33 ,rs:155.5 4.224.53  
0180 3a 33 33 39 39 33 0a 66 69 6c 65 6e 61 6d 65 3a ,:33993.f ilename:  
0190 75 70 6c 6f 61 64 65 64 5f 64 5f 61 72 63 68 69 ,uploaded _d archi  
01a0 76 6f 5f 34 0a 73 69 7a 65 3a 36 38 34 35 38 33 ,vo_4.siz e:684583  
01b0 33 0a 69 61 73 68 3a 35 38 37 65 38 64 35 39 3a ,3-hash:5 87e9d501  
01c0 31 61 39 62 30 65 36 39 32 33 32 34 33 36 33 31 ,1a8b0e09 23249631  
01d0 33 32 34 35 65 62 38 61 65 36 65 39 38 33 38 0a ,3245eb8a e6e9038  
01e0 73 65 72 76 65 72 73 3a 31 35 35 2e 35 34 2e 32 ,servers: 155.54.2  
01f0 32 34 2e 35 33 3a 33 33 39 39 33 0a 0a ,24.53:33 993..
```


Envía Mensaje Download

*eth0

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	155.54.224.55	155.54.224.54	UDP	81	33199 → 6868 Len=39
2	0.006363408	155.54.224.54	155.54.224.55	UDP	128	6868 → 33199 Len=86
3	0.012558393	ASUSTekCOMPU_d5:0a:...	Broadcast	ARP	60	Who has 155.54.224.53? Tell 155.54.224.55
4	0.744776242	Cisco_3b:b4:14	Spanning-tree-(for-...	STP	119	MST, Root = 16384/0/7c:69:f6:66:fd:80 Cost = 2000 Port = 0x8014

Frame 1: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface 0

Ethernet II, Src: ASUSTekCOMPU_d5:0a:51 (2c:4d:54:d5:0a:51), Dst: ASUSTekCOMPU_d5:0a:51 (2c:4d:54:d5:0a:51)

Internet Protocol Version 4, Src: 155.54.224.55, Dst: 155.54.224.54

User Datagram Protocol, Src Port: 33199, Dst Port: 6868

Data (39 bytes)

0000 2c 4d 54 d5 1b 58 2c 4d 54 d5 0a 51 08 00 45 00 ,MT..X.M T..Q..E..

0010 00 43 ab 71 40 00 40 11 98 5d 9b 36 e0 37 9b 36 ,C.q@.].6.7.6

0020 e0 36 81 af 1a d4 00 2f 3a db 6f 70 65 72 61 74 ,6.....:operat

0030 69 6f 6e 3a 64 6f 77 6e 6c 6f 61 64 0a 66 69 6c ion:down load:fil

0040 65 6e 61 6d 65 3a 61 72 63 68 69 76 6f 5f 32 0a ename:ar chivo_2.

0050 0a

Data: 6f7065726174696f6e3a646f776e6c6f61640a6666696c656e616d653a6172636869766f5f320a0a [Length: 39]

Recibe ServersSharingThisFile

*eth0

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	155.54.224.55	155.54.224.54	UDP	81	33199 → 6868 Len=39
2	0.006363408	155.54.224.54	155.54.224.55	UDP	128	6868 → 33199 Len=86
3	0.012558393	ASUSTekCOMPU_d5:0a:...	Broadcast	ARP	60	Who has 155.54.224.53? Tell 155.54.224.55
4	0.744776242	Cisco_3b:b4:14	Spanning-tree-(for-...	STP	119	MST, Root = 16384/0/7c:69:f6:66:fd:80 Cost = 2000 Port = 0x8014

Frame 2: 128 bytes on wire (1024 bits), 128 bytes captured (1024 bits) on interface 0

Ethernet II, Src: ASUSTekCOMPU_d5:1b:58 (2c:4d:54:d5:1b:58), Dst: ASUSTekCOMPU_d5:1b:58 (2c:4d:54:d5:1b:58)

Internet Protocol Version 4, Src: 155.54.224.54, Dst: 155.54.224.55

User Datagram Protocol, Src Port: 6868, Dst Port: 33199

Data (86 bytes)

0000 2c 4d 54 d5 0a 51 2c 4d 54 d5 1b 58 08 00 45 00 ,MT..Q.M T..X..E..

0010 00 72 c7 cb 40 00 40 11 7b d4 9b 36 e0 36 9b 36 ,r..@.@. (.6.6.6

0020 e0 37 1a d4 81 af 00 5e f7 4a 6f 70 65 72 61 74 ,7.....^Joperat

0030 69 6f 6e 3a 73 65 72 76 65 72 73 53 68 61 72 69 ion:serve rsShari

0040 6e 67 54 08 69 73 46 69 6c 65 0a 66 69 6c 65 6e ngThisFi le:filen

0050 61 6d 65 3a 61 72 63 68 69 76 6f 5f 32 2e 64 6f ame:arch ivo_2.do

0060 63 78 0a 73 65 72 76 65 72 3a 2f 31 35 35 2e 35 cx-serve r:/155.5

0070 34 2e 32 32 34 2e 35 33 3a 33 33 39 39 33 0a 0a 4,224.53 :33993...

Data: 6f7065726174696f6e3a7365727665727353686172696e675468697346696c656e616d653a6172636869766f5f322e646f342e3232342e35333a33333939330a0a [Length: 86]

7. Conclusiones

El desarrollo del sistema nanoFilesP2P ha permitido comprender e implementar conceptos fundamentales de redes de computadores, como la comunicación mediante sockets UDP y TCP, el diseño de protocolos de mensajería ASCII, y la coordinación entre múltiples peers en una arquitectura distribuida tipo P2P.

Durante el proyecto, se ha logrado construir una infraestructura funcional que permite a distintos usuarios compartir, descargar y subir archivos de forma eficiente y descentralizada, con la ayuda de un servidor directorio centralizado que actúa como punto de coordinación.

A lo largo del proceso, se han abordado múltiples retos, entre ellos: el diseño de protocolos personalizados, para lo cual fue necesario definir y parsear mensajes ASCII bien estructurados para intercambiar información entre clientes, servidores y el directorio; y la gestión de sockets y concurrencia, para lo que se trabajó con sockets TCP con el objetivo de establecer múltiples conexiones en paralelo.

Además, se han implementado mejoras relevantes al diseño original que amplían su funcionalidad y robustez. Gracias a estas mejoras, el sistema es más dinámico, realista y útil en un entorno simulado de compartición de ficheros. Además, el trabajo ha fomentado el pensamiento modular, la reutilización de código y el uso de estructuras como mapas para gestionar asociaciones clave entre direcciones y ficheros.

En resumen, *nanoFilesP2P* ha sido una experiencia muy útil para comprender cómo funciona un sistema donde varios usuarios pueden compartir archivos entre sí. A lo largo del proyecto se han aplicado ideas clave sobre cómo se comunican los programas entre diferentes ordenadores, y se ha aprendido a organizar bien el código para que todo funcione de forma coordinada y fiable. Además, las mejoras añadidas han permitido construir un sistema más completo, flexible y cercano a cómo funcionan realmente este tipo de aplicaciones.