

**DOCUMENTO DE DISEÑO**

**PROYECTO NANOFILES**

**Asignatura: Redes de Computadores**

**Autor: Juan Rondán Ramos. DNI: 48854653P**

**Autor: Luis Molina Llamos. DNI: 55142089A**

**Profesor: Juan José Pujante Moreno**

**Convocatoria de Mayo 2025**

**Facultad de Informática**

**Universidad de Murcia**

Índice

Contenido

[Introducción. 3](#_Toc197181709)

[Protocolos diseñados 4](#_Toc197181710)

[Directorio 4](#_Toc197181711)

[*Formato de los mensajes y ejemplos de los mismos* 4](#_Toc197181712)

[*Autómatas cliente y servidor* 9](#_Toc197181713)

[Peer To Peer 10](#_Toc197181714)

[*Formato de los mensajes y ejemplos de los mismos* 10](#_Toc197181715)

[*Autómatas cliente y servidor* 13](#_Toc197181716)

[Mejoras implementadas 14](#_Toc197181717)

[Capturas de pantalla Wireshark 15](#_Toc197181718)

[Conclusiones 18](#_Toc197181719)

# Introducción.

El sistema NanoFiles está formado por un servidor de directorio (programa Directory) y un conjunto de peers o pares (programa NanoFiles), que se comunican entre sí de la siguiente forma:

* Por un lado, la comunicación entre cada peer de NanoFiles y el servidor de directorio se rige por el modelo cliente-servidor. Un peer actúa como cliente del directorio para consultar los ficheros que pueden ser descargados de otros peers, publicar los ficheros que quiere compartir con el resto de pares y obtener los servidores que comparten un determinado fichero.
* Por otro lado, el modelo de comunicación entre pares de NanoFiles es peer-to-peer (P2P).
  + Cuando un peer actúa como cliente de otro peer servidor, el cliente puede consultar los ficheros disponibles en el servidor y descargar aquellos fragmentos (chunks) de un fichero determinado que solicite.
  + De manera complementaria, un peer puede convertirse a petición del usuario en servidor de ficheros, de forma que escuche en un puerto determinado en espera de que otros peers se conecten para solicitarle fragmentos de los ficheros que está compartiendo.

# Protocolos diseñados

## Directorio

### Formato de los mensajes y ejemplos de los mismos

Para el intercambio de mensajes y comunicación entre el directorio y un peer servidor, nos centraremos en el formato campo:valor. Este formato utiliza delimitadores basados en texto y facilita la legibilidad de los mensajes. El protocolo de comunicación entre Directorio y peer cliente se establece sobre un canal no confiable UDP. Los mensajes ASCII diseñados para este intercambio de información son los siguientes:

* Mensaje: ping

Sentido de la comunicación: Peer cliente -> Directorio

Descripción: El peer cliente contacta con el servidor de directorio para comprobar que está a la escucha y que utiliza un protocolo compatible con el suyo.

* Mensaje: pingok

Sentido de la comunicación: Directorio -> Peer cliente

Descripción: El Directorio responde al peer cliente informándole de que está a la escucha y de que usa un protocolo compatible.

* Mensaje: pingError

Sentido de la comunicación: Directorio -> Peer cliente

Descripción: El Directorio responde al peer cliente informándole de que ha ocurrido un error y no se ha podido establecer la conexión.

* Ejemplo de intercambio de mensaje:
* Caso de acierto:

Peer -> Directorio

operation: ping

protocol: 48854653P

Directorio -> Peer

operation: pingok

* Caso de fallo:

Peer -> Directorio

operation: ping

protocol: 48854653P

Directorio -> Peer

operation: pingError

* Mensaje: serve

Sentido de la comunicación: Peer cliente -> Directorio

Descripción: El peer cliente lanza una solicitud para publicar su lista de ficheros compartidos, presentes en la carpeta nf-shared.

* Mensaje: serveok

Sentido de la comunicación: Directorio -> Peer cliente

Descripción: El Directorio responde al peer cliente indicándole que sus ficheros compartidos en la carpeta nf-shared se han publicado con éxito.

* Mensaje: serveError

Sentido de la comunicación: Directorio -> Peer cliente

Descripción: El Directorio responde al peer cliente indicándole que no hay ficheros compartidos en la carpeta nf-shared y que por tanto no se puede publicar nada.

* Ejemplo de intercambio de mensaje:
* Caso de acierto:

Peer -> Directorio

operation: serve

port: 57611

filecount: 2

filename: prueba.txt

size: 150

hash: ed1f33982fb019e3ab924a635227c290dceddbd6

filename: otro.txt

size: 61

hash: 331fb8c612206a30c691eda742ef5cd760b74551

Directorio -> Peer

operation: serveok

* Caso de fallo:

Peer -> Directorio

operation: serve

port: 57611

filecount: 0

Directorio -> Peer

operation: serveError

* Mensaje: filelist

Sentido de la comunicación: Peer cliente -> directorio

Descripción: El peer cliente lanza una solicitud para mostrar la lista de ficheros que han sido publicados al Directorio

* Mensaje: filelistok

Sentido de la comunicación: Directorio -> Peer cliente

Descripción: El Directorio responde al peer mostrando la lista de ficheros que hay publicados.

* Mensaje: filelistError

Sentido de la comunicación: Directorio -> Peer cliente

Descripción: El Directorio responde al peer cliente indicando que ningún fichero ha sido publicado.

* Ejemplo de intercambio de mensaje:
* Caso de acierto:

Peer -> Directorio

operation: filelist

Directorio -> Peer

operation: filelistok

filecount: 2

filename: prueba.txt

size: 150

hash: ed1f33982fb019e3ab924a635227c290dceddbd6

filename: otro.txt

size: 61

hash: 331fb8c612206a30c691eda742ef5cd760b74551

* Caso de fallo:

Peer -> Directorio

operation: filelist

Directorio -> Peer

operation: filelistError

* Mensaje: myfiles

Sentido de la comunicación: Local

Descripción: El peer muestra en pantalla los ficheros que tiene localmente en la carpeta nf-shared, indicando nombre, tamaño y hash.

* Ejemplo de intercambio de mensaje:
* Caso de tener ficheros en la carpeta nf-shared:

Peer -> Directorio

operation: myfiles

Directorio -> Peer

List of files in local folder:

Name Size Hash

prueba.txt 10 ed1f33982fb019e3ab924

foto.jpg 485918 25312d3c18c25f64edf70d

hola.txt 61 91eda742ef5cd760b74551

* Caso de no tener ficheros en la carpeta nf-shared:

Peer -> Directorio

operation: myfiles

Directorio -> Peer

List of files in local folder:

Name Size Hash

(mostramos la tabla vacía, no hay respuesta de acierto ni de error)

* Mensaje: download <subcadena fichero a buscar> <nombre archivo descargado>

Sentido de la comunicación: Peer cliente -> Directorio

Descripción: El peer cliente solicita descargar un fichero de los existentes en el servidor de Directorio, cuyo nombre encaje con la subcadena introducida en el primer parámetro

* Mensaje: downloadok

Sentido de la comunicación: Directorio -> Peer cliente

Descripción: El Directorio indica que existe un archivo con la subcadena indicada, y que la descarga ha sido realizada con éxito.

* Mensaje: downloadError

Sentido de la comunicación: Directorio -> Peer cliente

Descripción: El Directorio indica que no existe un fichero con la subcadena recibida o que la propia descarga ha fallado.

* Ejemplo de intercambio de mensaje:
* Caso de acierto:

Peer -> Directorio

operation: download prueba.txt new\_name.txt

Directorio -> Peer

operation: downloadok

* Caso de fallo:

Peer -> Directorio

operation: download text.txt nombre.txt

Directorio -> Peer

operation: downloadError

* Mensaje: help

Sentido de la comunicación: Local

Descripción: Muestra ayuda sobre las órdenes soportadas

* Ejemplo de intercambio de mensaje:

Peer -> Directorio

operation: help

Directorio -> Peer

List of commands:

quit -- quit the application

ping -- ping directory to check protocol compatibility

filelist -- show list of files tracked by the directory

myfiles -- show contents of local folder (files that may be served)

serve -- run file server and publish served files to directory

download -- download file from all available server(s)

help -- shows this information

* Mensaje: quit

Sentido de la comunicación: Local

Descripción: Sale de la aplicación

Peer -> Directorio

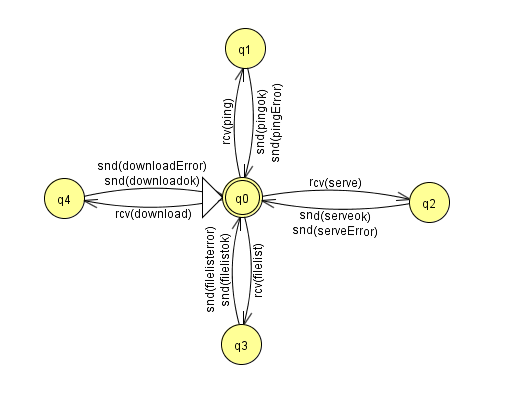
operation: quit

Directorio -> Peer

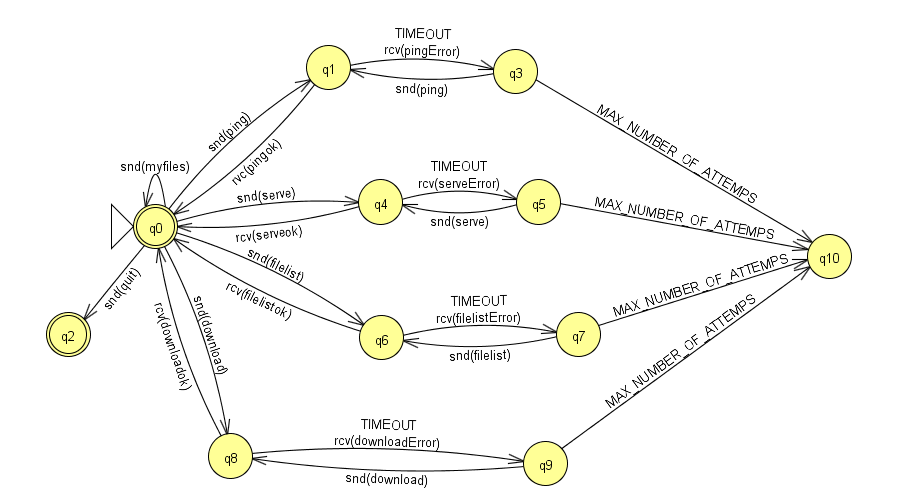
Bye.

### Autómatas cliente y servidor

* Autómata Directorio:



* Autómata Peer cliente



## Peer To Peer

### Formato de los mensajes y ejemplos de los mismos

Para llevar a cabo la transferencia de ficheros entre peers, se han implementado mensajes binarios multiformato que son intercambiados entre un peer servidor de ficheros y un peer cliente que desea descargar los ficheros disponibles. El protocolo de comunicación entre cliente y servidor de ficheros se establece sobre un canal confiable TCP. Los mensajes binarios diseñados son los siguientes:

* Mensaje: FILE\_NOT\_FOUND

Sentido de la comunicación: Peer servidor -> Peer cliente

Descripción: El servidor de ficheros responde con este mensaje cuando no encuentra un fichero que coincida con la subcadena solicitada o cuando el número de chunk solicitado está fuera de rango respecto al tamaño del fichero.

Opcode: 0x01

Formato del mensaje:

|  |
| --- |
| Opcode |
| 1 byte |

* Mensaje: FILE\_INFO\_REQUEST

Sentido de la comunicación: Peer cliente -> Peer servidor

Descripción: El cliente solicita al servidor de ficheros información detallada (nombre, tamaño y hash) de un fichero cuyo nombre contenga la subcadena indicada. El servidor responderá con los datos del primer fichero coincidente.

Opcode: 0x04

Formato del mensaje:

|  |  |
| --- | --- |
| Opcode | Subcadena |
| 1 byte | n bytes |

* Mensaje: FILE\_INFO\_RESPONSE

Sentido de la comunicación: Peer servidor -> Peer cliente

Descripción: El servidor de ficheros responde al FILE\_INFO\_REQUEST proporcionando el nombre, el tamaño y el hash del primer fichero cuyo nombre contiene la subcadena enviada por el cliente.

Opcode: 0x05

Formato del mensaje:

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Nombre fichero | Tamaño | Hash |
| 1 byte | n bytes | 8 bytes | m bytes |

* Mensaje: GET\_CHUNK

Sentido de la comunicación: Peer cliente -> Peer Servidor

Descripción: El cliente solicita a un peer servidor un fragmento (chunk) específico de un fichero determinado. A partir de la información recibida en el FILE\_INFO\_RESPONSE (tamaño del fichero) y el número de peers que tengan el fichero, el cliente calcula la división en chunks. Luego envía un mensaje GET\_CHUNK a cada peer, indicando el número de chunk y el tamaño que se espera recibir. Cada peer servidor recibe un GET\_CHUNK único, indicando qué parte exacta del fichero debe enviar.

Opcode: 0x02

Formato del mensaje:

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Nombre fichero | Número de chunk | Tamaño de chunk |
| 1 byte | n bytes | 4 bytes | m bytes |

* Mensaje: SEND\_CHUNK

Sentido de la comunicación: Peer servidor -> Peer cliente

Descripción: El servidor responde enviando el fragmento solicitado en el GET\_CHUNK. Al recibir el tamaño del chunk y el número de chunk, el servidor calcula localmente el offset en el fichero, lee el número de bytes solicitados y envía el fragmento binario como campo de datos en el mensaje.

Formato del mensaje:

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Nombre fichero | Número de chunk | Datos |
| 1 byte | n bytes | 4 bytes | m bytes |

* Ejemplo de intercambio de mensajes:
* Caso de acierto:

Cliente -> Servidor

Mensaje: FILE\_INFO\_REQUEST

Se solicita información sobre el fichero cuyo nombre contiene una subcadena dada.

Servidor -> Cliente

Mensaje: FILE\_INFO\_RESPONSE

Responde con el nombre exacto, tamaño y hash del fichero.

Localmente en el cliente:

Calcula el tamaño del chunk (chunkSize).

Divide el fichero en chunks y asigna un chunk a cada peer disponible.

Cliente -> Servidor

Mensaje: GET\_CHUNK (uno por cada peer servidor disponible).

Solicita el chunk específico al servidor, indicándole número de chunk y tamaño.

Localmente en el servidor:

Calcula el offset, y la cantidad exacta de bytes que debe leer.

Servidor -> Cliente:

Mensaje: SEND\_CHUNK (en respuesta individual a cada GET\_CHUNK).

Responde enviando el fragmento binario del fichero correspondiente.

Cliente:

Ensambla los chunks recibidos y verifica la integridad final, comparando el hash calculado con el esperado.

* Caso de fallo

Cliente -> Servidor

Mensaje: FILE\_INFO\_REQUEST

Se solicita información sobre el fichero cuyo nombre contiene una subcadena dada.

Servidor -> Cliente

Mensaje: FILE\_NOT\_FOUND

No se ha encontrado un fichero cuyo nombre tenga coincidencia con la subcadena recibida.

O bien durante la propia descarga:

Cliente -> Servidor:

Mensaje: GET\_CHUNK

Se solicita un chunk que no existe, o un contenido de un fichero que ha sido modificado.

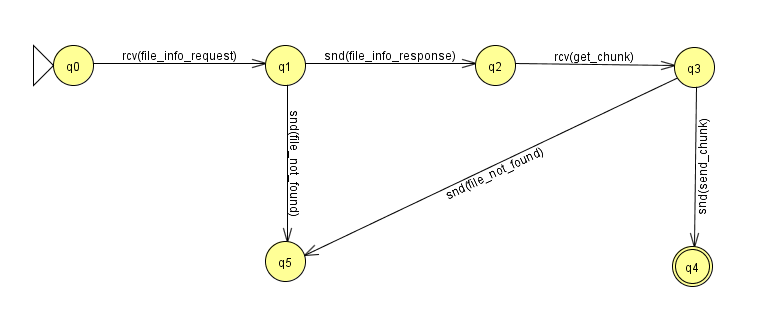
Servidor -> Cliente:

Mensaje: FILE\_NOT\_FOUND

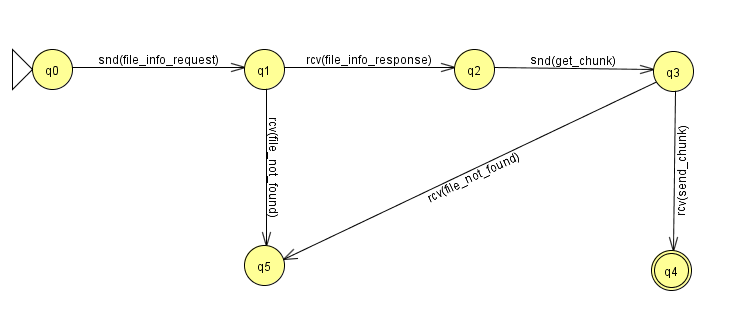
El servidor detecta el fallo, y responde con error.

### Autómatas cliente y servidor

* Autómata Peer servidor de ficheros:



* Autómata Peer cliente de ficheros:



# Mejoras implementadas

* Comando serve con puerto efímero:

Se ha modificado el comportamiento del servidor de ficheros (NFServer) para que, en lugar de escuchar en un puerto fijo, por ejemplo, el 10.000, utilice un puerto efímero asignado automáticamente por el sistema operativo.

Para ello en la clase NFServer se ha sustituido el valor de la constante PORT de 10000 a 0, del modo:

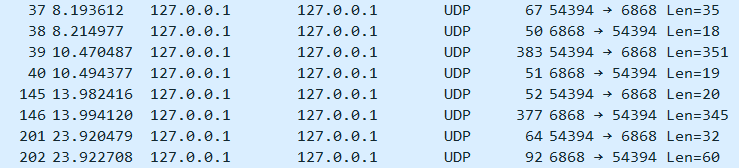
public static final int PORT = 0;

En Java, cuando se crea un ServerSocket con puerto 0, el sistema asigna dinámicamente un puerto libre disponible.

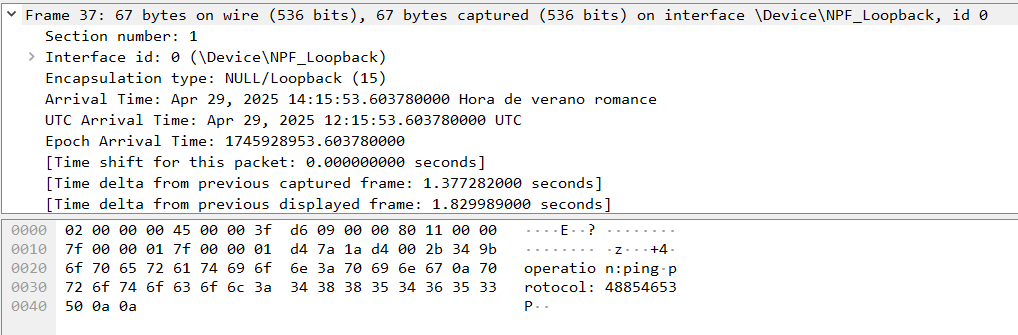
Para la consulta del puerto asignado tras la creación del servidor, se ha implementado el método getListeningPort(), que permite recuperar el número de puerto que está utilizando el servidor de ficheros.

# Capturas de pantalla Wireshark

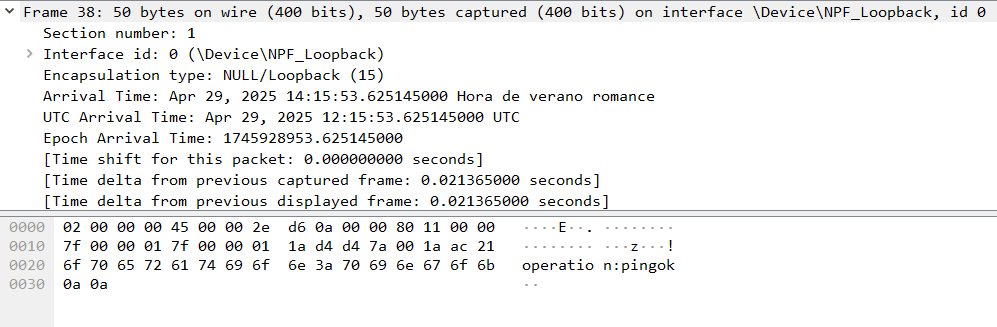
* Intercambio completo



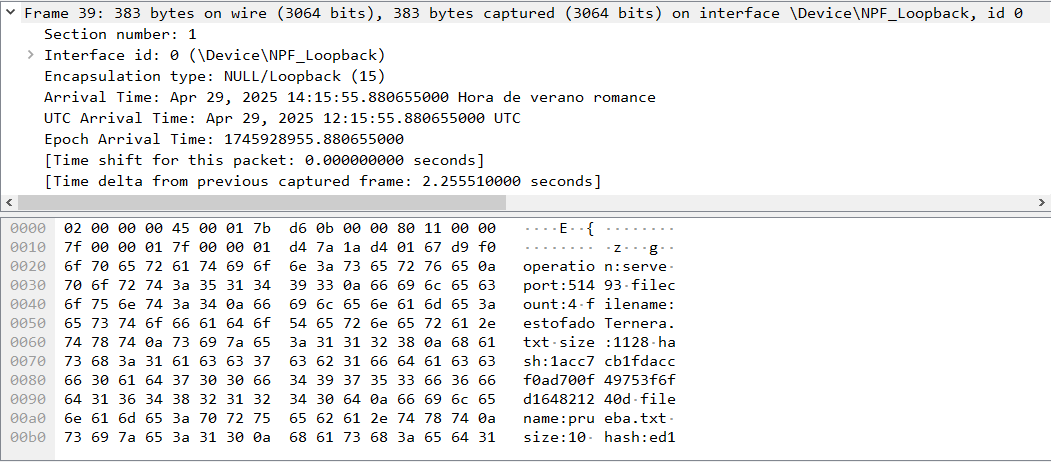
* Frame 37: ping



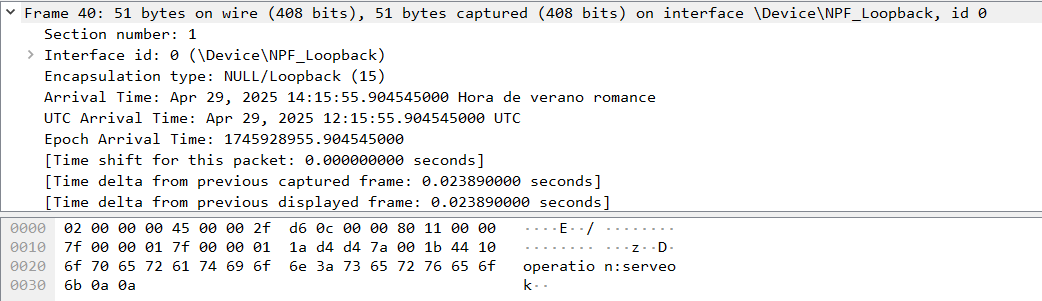
* Frame 38: pingok



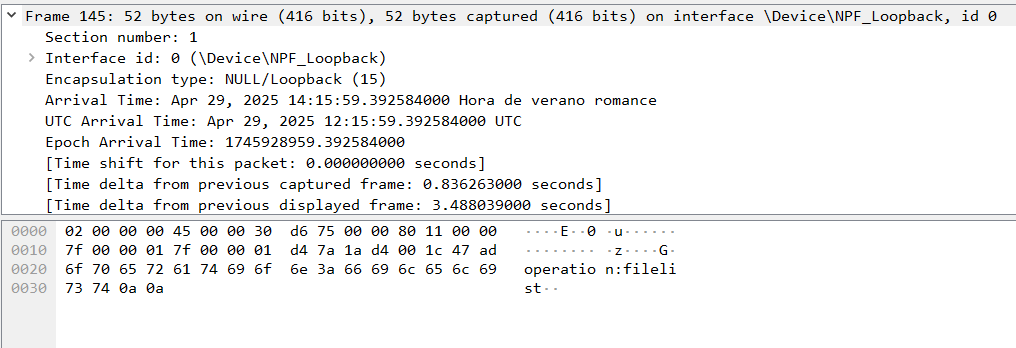
* Frame 39: serve



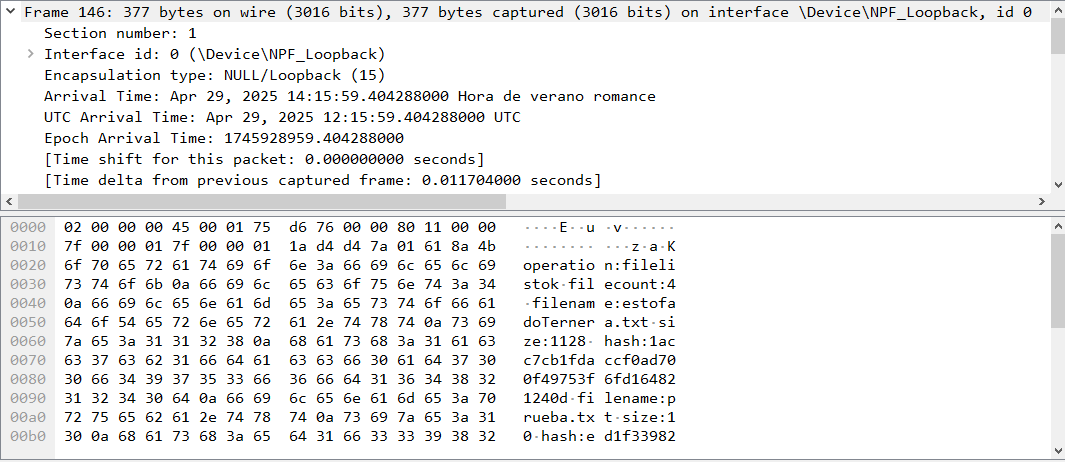
* Frame 40: serveok



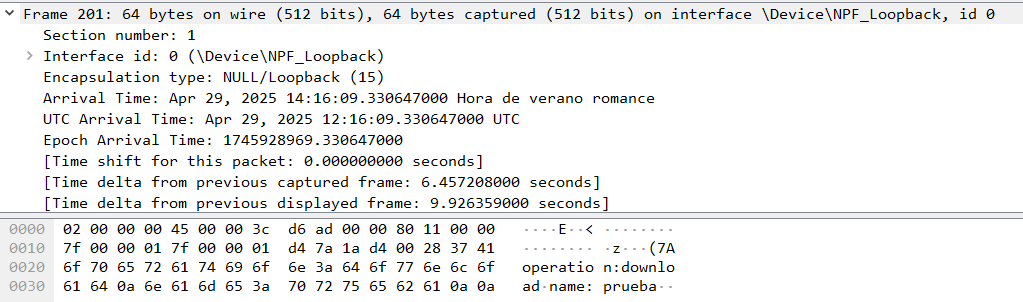
* Frame 145: filelist



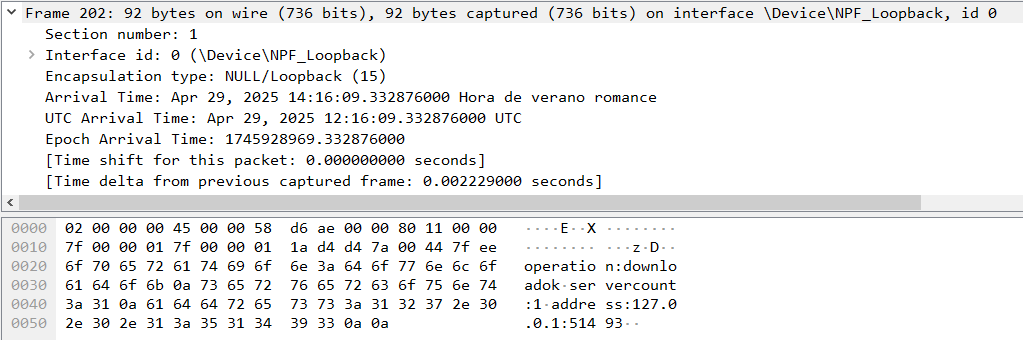
* Frame 146: filelistok



* Frame 201: download



* Frame 202: downloadok



# Conclusiones

Este proyecto ha sido, sin duda, uno de los mayores retos a los que nos hemos enfrentado. Al principio, la práctica parecía inmanejable: múltiples capas de comunicación, diferentes protocolos, servidores concurrentes, sockets, hilos… era difícil saber por donde empezar. La sensación inicial era de estar completamente perdido, sin una visión clara de cómo encajar entre sí todas las piezas del sistema.

Tras muchas horas frente al ordenador, todo comenzó poco a poco a tomar forma. El sistema fue cobrando sentido a medida que se implementaban las distintas partes y se veían los primeros resultados. Uno de los mayores desafíos fue la implementación de la comunicación TCP entre peers, en particular el proceso de descarga de ficheros. Diseñar un protocolo robusto, dividir los archivos en chunks, gestionar múltiples conexiones y asegurar la integridad de los datos requirió un gran esfuerzo, además de muchas iteraciones y pruebas. La depuración de errores en tiempo de ejecución y la revisión constante de que todo funcionara correctamente, también supuso un reto constante durante el desarrollo.

En resumen, ha sido una práctica tan compleja como gratificante. Más allá del código, nos ha enseñado a ser más pacientes, a planificar mejor, a depurar con cabeza y, sobre todo, a confiar en que, aunque algo al principio parezca imposible, con trabajo y persistencia puede salir adelante.