



(75.43) Introducción a los sistemas distribuidos

Grupo A10

Trabajo Práctico

Segundo cuatrimestre - 2023

Alumno	Padrón	Mail
Agustín D. Palmeira	90.856	apalmeira@fi.uba.ar
Alejandro Paff	103376	apaff@fi.uba.ar
Alejo Acevedo	99146	aacevedo@fi.uba.ar
Federico Cavazzoli	98533	fcavazzoli@fi.uba.ar

Índice

Introducción.....	3
Hipótesis y suposiciones realizadas.....	3
Implementación.....	3
Mininet.....	3
Prerrequisitos de instalación en Ubuntu para mininet.....	4
Instalación de mininet en Ubuntu.....	4
Cómo correr la topología.....	4
Cómo utilizar mininet.....	4
Logs creados.....	5
Diagrama.....	5
Servidor.....	5
Interfaz del servidor.....	5
Cliente.....	6
Interfaz del cliente.....	6
Upload.....	6
Download.....	6
Pruebas.....	7
Ejecución de pruebas.....	7
Wireshark.....	8
Tests unitarios.....	8
Prerrequisitos de instalación en Ubuntu.....	9
Cómo ejecutar los tests.....	9
Preguntas.....	9
1. Describa la arquitectura cliente-servidor.....	9
2. ¿Cuál es la función de un protocolo de capa de aplicación?.....	10
3. Detalle el protocolo de aplicación desarrollado en este trabajo.....	10
4. La capa de transporte del stack TCP/IP ofrece dos protocolos: TCP y UDP. ¿Qué servicios proveen dichos protocolos? ¿Cuáles son sus características? ¿Cuándo es apropiado utilizar cada uno?.....	12
Dificultades encontradas.....	13
Conclusión.....	13
Bibliografía.....	13

Introducción

El propósito de este informe es proporcionar una descripción detallada del proceso de desarrollo de la aplicación de red. A lo largo de este documento, se presentarán los aspectos clave del proyecto en el contexto del aprendizaje relacionado con sistemas distribuidos y protocolos de la capa de aplicación y transporte.

Hipótesis y suposiciones realizadas

A continuación se detallan las hipótesis más relevantes asumidas en el trabajo:

- Los hosts, tanto los clientes como el servidor, disponen de espacio suficiente para el almacenamiento de los archivos que se envíen.
- Para el control de la transmisión de datos, se pueden emplear dos estrategias diferentes: "stop and wait" o "selective repeat". No obstante, es fundamental destacar que la estrategia elegida debe ser uniforme en todos los clientes participantes.
- Es importante tener en cuenta que, si un cliente está actualmente subiendo un archivo, otro cliente no podrá descargar ese mismo archivo hasta que la subida se haya completado en su totalidad. Esto es esencial para evitar la pérdida de paquetes y la eventual expiración de tiempos de espera ("timeout").
- En situaciones donde se realice una carga ("upload") de un archivo con un nombre que ya existe en el sistema, el archivo existente será reemplazado por el nuevo archivo cargado.
- En situaciones donde se realice una descarga ("download") de un archivo con un nombre que ya existe en el sistema, el archivo nuevo será añadido al archivo existente, es decir, no se sobrescribirá como en el caso de una carga.

Implementación

En esta sección, se proporcionará una descripción detallada de la implementación general realizada, abordando tanto la configuración de la topología como la lógica detrás de las operaciones de carga ("upload") y descarga ("download") de archivos, en función de la estrategia seleccionada.

Mininet

Para diseñar la topología, se desarrollaron dos scripts que simplifican la creación y ejecución de la configuración deseada en un entorno Linux con un solo comando.

El primer parámetro que se debe especificar es la cantidad de hosts, incluyendo el servidor, mientras que el segundo parámetro determina el porcentaje de pérdida de paquetes del servidor únicamente.

La decisión de limitar el número máximo de hosts a diez se basa en consideraciones de recursos. Esto se debe a que cada host en Mininet se ejecuta como una instancia separada de una máquina virtual, lo que implica que su configuración puede aumentar la carga de recursos, como la CPU y la memoria. En otras palabras, la administración de recursos podría sobrecargarse si se crean demasiados hosts.

Para ilustrar esta limitación, en nuestras pruebas con una computadora portátil que tiene 4 GB de RAM, observamos que el sistema dejaba de funcionar correctamente cuando se superaba la cifra de diez hosts.

Prerrequisitos de instalación en Ubuntu para mininet

Se deben instalar las siguientes dependencias mediante la línea de comandos:

```
...  
> sudo apt-get -y install xterm  
...
```

Instalación de mininet en Ubuntu

```
...  
> sudo apt-get install mininet  
...
```

Cómo correr la topología

Desde la consola ejecutar (por ejemplo con 2 hosts y con un 10% de pérdida de paquetes):

```
...  
python3 create_topology_from_scratch.py 2 10  
...
```

El primer parámetro es referido a la cantidad de hosts, y tiene que ser un número positivo, mayor a 2 y menor a 10. Debe ser mayor a 2 para que puedan realizarse transferencias.

Cómo utilizar mininet

Para correr la consola de un host se debe ejecutar el siguiente comando:

```
...  
> h1 xterm &  
...
```

Logs creados

Se crearán dos archivos con los sobre la cantidad de hosts y porcentajes de pérdidas de paquetes definidos.

Nota: Para más información, ver el readme respectivo dentro del proyecto.

Diagrama

A continuación se detalla el diagrama general de la topología. Nótese que la cantidad de hosts será entre dos y diez, siendo uno de ellos el que actuará como servidor y el resto como clientes.

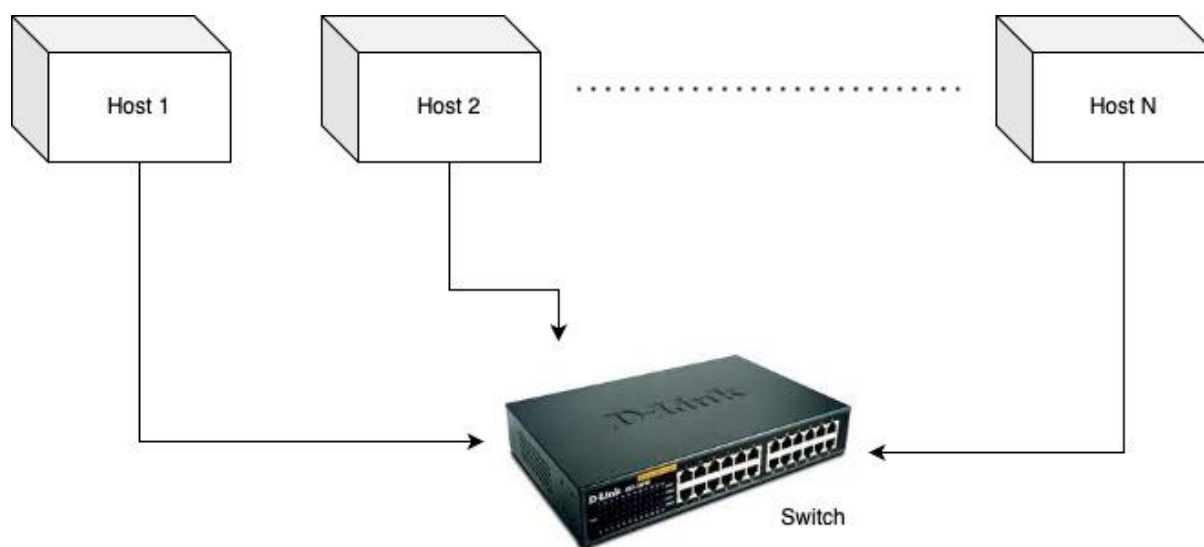


Fig. 1 Topologia de mininet

Servidor

Interfaz del servidor

El servidor provee el servicio de almacenamiento y descarga de archivos. Por consola se debe utilizar el siguiente comando:

...

> python start - server -h

usage : start - server [-h] [-v | -q] [-H ADDR] [-p PORT] [-s DIRPATH]

<command description >

optional arguments :

-h, --help show this help message and exit

-v, --verbose increase output verbosity

-q, --quiet decrease output verbosity

-H, --host service IP address

-p, --port service port

-s, --storage storage dir path

Cliente

Interfaz del cliente

La funcionalidad del cliente se divide en dos aplicaciones de línea de comandos: upload y download.

Upload

Esta funcionalidad se utiliza para enviar un archivo al servidor y que sea guardado con un nombre especificado. Por consola se debe utilizar el siguiente comando:

```
...  
> python upload -h  
usage : upload [-h] [-v | -q] [-H ADDR ] [-p PORT ] [-s FILEPATH ] [-n FILENAME ]  
<command description >  
optional arguments :  
-h, --help show this help message and exit  
-v, --verbose increase output verbosity  
-q, --quiet decrease output verbosity  
-H, --host server IP address  
-p, --port server port  
-s, --src source file path  
-n, --name file name  
...
```

Download

Esta funcionalidad se utiliza para enviar un archivo al servidor y que sea guardado con un nombre especificado. Por consola se debe utilizar el siguiente comando:

```
...  
> python download -h  
usage : download [-h] [-v | -q] [-H ADDR ] [-p PORT ] [-d FILEPATH ] [-n FILENAME ]  
<command description >  
optional arguments :  
-h, --help show this help message and exit  
-v, --verbose increase output verbosity  
-q, --quiet decrease output verbosity  
-H, --host server IP address  
-p, --port server port  
-d, --dst destination file path  
-n, --name file name  
...
```

Pruebas

En esta sección se detallan los tipos de pruebas realizadas.

Ejecución de pruebas

Se realizaron pruebas con diferentes archivos, de hasta 250mb y fueron satisfactorias.

A continuación, se deja como evidencia parte de la ejecución del upload de una imagen:

Ejecución del servidor:

```
[→ src git:(master) ✖ python3 start-server.py
Server started at localhost:5000
INFO      Server started
received new connection: b'handshake'
OPT: METADATA
SERVER received upload message
file_name: store/hi.png
SERVER received message: METADATA
_ payload:b'hi.png',
```

Ejecución del upload de un archivo llamado “*hi.png*” (no mostraremos todos los envíos paquetes ya que son más de 100):

```
[→ src git:(master) ✖ python3 upload.py -n hi.png
chunk_size: 1023
INFO      Client upload started
INFO      Connected to server
file_name_bytes: b'\x00\x01hi.png'
SENDING 0
ACK RECEIVED 0
SENDING 1
ACK RECEIVED 1
SENDING 2
ACK RECEIVED 2
SENDING 3
ACK RECEIVED 3
SENDING 4
ACK RECEIVED 4
```

Ejecución del download de un archivo llamado “*hi.png*”:

```
[→ src git:(master) ✖ python3 download.py -n hi.png
INFO      Client download started
INFO      Connected to server
file_name_bytes: b'\x00\x00hi.png'
SENDING 0
ACK RECEIVED 0
Download completed
```

Nótese que por default el protocolo utilizado es stop and wait.

Wireshark

Visto desde wireshark, podemos ver que se trata de la imagen en cuestión:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	41	56892 → 5000 Len=9
2	0.001590	127.0.0.1	127.0.0.1	UDP	50	54927 → 56892 Len=18
3	0.002019	127.0.0.1	127.0.0.1	UDP	52	56892 → 54927 Len=20
4	0.002200	127.0.0.1	127.0.0.1	UDP	44	54927 → 56892 Len=12
5	0.002384	127.0.0.1	127.0.0.1	UDP	1068	56892 → 54927 Len=1036
6	0.002635	127.0.0.1	127.0.0.1	UDP	44	54927 → 56892 Len=12
7	0.002809	127.0.0.1	127.0.0.1	UDP	1068	56892 → 54927 Len=1036
8	0.002877	127.0.0.1	127.0.0.1	UDP	44	54927 → 56892 Len=12
9	0.003113	127.0.0.1	127.0.0.1	UDP	1068	56892 → 54927 Len=1036
10	0.003175	127.0.0.1	127.0.0.1	UDP	44	54927 → 56892 Len=12
11	0.003313	127.0.0.1	127.0.0.1	UDP	1068	56892 → 54927 Len=1036
12	0.003370	127.0.0.1	127.0.0.1	UDP	44	54927 → 56892 Len=12
13	0.003539	127.0.0.1	127.0.0.1	UDP	1068	56892 → 54927 Len=1036
14	0.003590	127.0.0.1	127.0.0.1	UDP	44	54927 → 56892 Len=12
15	0.003732	127.0.0.1	127.0.0.1	UDP	1068	56892 → 54927 Len=1036
16	0.003829	127.0.0.1	127.0.0.1	UDP	44	54927 → 56892 Len=12
17	0.003964	127.0.0.1	127.0.0.1	UDP	1068	56892 → 54927 Len=1036
18	0.004020	127.0.0.1	127.0.0.1	UDP	44	54927 → 56892 Len=12
19	0.004154	127.0.0.1	127.0.0.1	UDP	1068	56892 → 54927 Len=1036
20	0.004206	127.0.0.1	127.0.0.1	UDP	44	54927 → 56892 Len=12
21	0.004342	127.0.0.1	127.0.0.1	UDP	1068	56892 → 54927 Len=1036
22	0.004394	127.0.0.1	127.0.0.1	UDP	44	54927 → 56892 Len=12
23	0.004538	127.0.0.1	127.0.0.1	UDP	1068	56892 → 54927 Len=1036
24	0.004596	127.0.0.1	127.0.0.1	UDP	44	54927 → 56892 Len=12

Frame 3: 52 bytes on wire (416 bits), 52 bytes captured (416 bits) on interface lo0, id 1

Section number: 1

Interface id: 0 (lo0)

Encapsulation type: NULL/Loopback (15)

Arrival Time: Oct 3, 2023 14:32:41.474636000 -03

[Time shift for this packet: 0.000000000 seconds]

Epoch Time: 1696354361.474636000 seconds

[Time delta from previous captured frame: 0.000429000 seconds]

[Time delta from previous displayed frame: 0.000429000 seconds]

[Time since reference or first frame: 0.002019000 seconds]

Frame Number: 3

Frame Length: 52 bytes (416 bits)

Capture Length: 52 bytes (416 bits)

0000 02 00 00 00 45 00 00 30 7c 46 00 00 40 11 00 00E..0|F..@...

0010 7f 00 00 01 7f 00 00 01 de 3c d6 8f 00 1c fe 2f<...../

0020 00 00 00 00 00 00 00 00 00 00 00 00 00 01 68 69hi

0030 2e 70 6e 67png

Por último, podemos apreciar que el archivo en cuestión es el correcto mediante md5, comparando los archivos de la carpeta /store y la carpeta /downloads:

```

[→ store git:(master) × ls
\hi.png
[→ store git:(master) × pwd
/Users/agustinpalmeira/Documents/Proyectos/facultad/intro/UDP-File-Transfer/src/store
[→ store git:(master) × md5 hi.png
MD5 (hi.png) = 3f6918035805933fb4cb7bb2975ec92e
[→ store git:(master) × ..
[→ src git:(master) × cd downloads
[→ downloads git:(master) × md5 hi.png
MD5 (hi.png) = 3f6918035805933fb4cb7bb2975ec92e
[→ downloads git:(master) × pwd
/Users/agustinpalmeira/Documents/Proyectos/facultad/intro/UDP-File-Transfer/src/downloads

```

Tests unitarios

Se llevaron a cabo pruebas unitarias en ciertas clases del proyecto con el objetivo de garantizar el comportamiento actual así como la robustez frente a posibles modificaciones en dichas clases.

Prerrequisitos de instalación en Ubuntu

Como requisito previo a la ejecución, se debe instalar la dependencia pytest. Por consola se debe utilizar el siguiente comando:

```
...  
> pip3 install pytest  
...
```

Cómo ejecutar los tests

Luego de haber instalado pytest, por consola se debe utilizar el siguiente comando:

```
...  
> python3 -m unittest  
...
```

Como resultado, aparecerán los logs correspondientes en la consola sobre todos los tests realizados.

Preguntas

En esta sección se responderá el cuestionario solicitado teórico y también específico del protocolo creado para el presente trabajo.

1. Describa la arquitectura cliente-servidor

La arquitectura Cliente-Servidor, donde siempre hay un host llamado servidor, con una IP address conocida, escucha peticiones de otros hosts llamados clientes, y va respondiendo cuando uno de estos host clientes se conecta enviando un mensaje. Esta arquitectura permite una distribución de tareas clara y eficiente, ya que los servidores se especializan en proporcionar servicios específicos, y los clientes se encargan de utilizar esos servicios.

Es importante destacar que los hosts clientes no se comunican directamente entre sí, sino que interactúan exclusivamente con el servidor. Si existen muchos clientes, se tendrá que contar con un data center para no saturar el host servidor

2. ¿Cuál es la función de un protocolo de capa de aplicación?

Las funciones principales de un protocolo de capa de aplicación son:

- a. Facilitar la comunicación entre aplicaciones en diferentes dispositivos de red mediante una interfaz de software llamada socket.
- b. Asegurar que los datos se transmitan de manera eficiente y segura.

Es la encargada de permitir que las aplicaciones se puedan comunicar entre sí, abstrayéndose de las diferencias en su estructura o formato de los datos. Establece la conexión entre las aplicaciones y asegura que los datos se transmitan de manera **segura y confiable**.

Además, también se encarga de otras funciones, como las siguientes:

- a. Controlar la velocidad a la que los datos se transmiten (para evitar la sobrecarga de una aplicación).
- b. Permitir que varias aplicaciones se comuniquen a través de la misma conexión de red.
- c. En esta capa el intercambio de mensajes se hace entre dos end-systems por medio de paquetes siguiendo un protocolo.

3. Detalle el protocolo de aplicación desarrollado en este trabajo.

En este trabajo, se ha desarrollado un protocolo de aplicación para facilitar la transferencia de archivos entre un cliente y un servidor. Este protocolo se basa en el uso de la capa de transporte de UDP (User Datagram Protocol) y se enfoca en la implementación de dos variantes de transferencia de datos confiable: Stop & Wait y Selective Repeat.

A continuación se enlistan los aspectos clave del TP:

- Comunicación Cliente-Servidor
- Protocolo RDT (Reliable Data Transfer)
- Transferencia de Archivos
- Condiciones de Error
- Interfaz de Línea de Comandos
- Log de acciones y estados
- Simulación de pérdida de paquetes

A continuación, se puede observar un diagrama de secuencia del upload de un archivo:

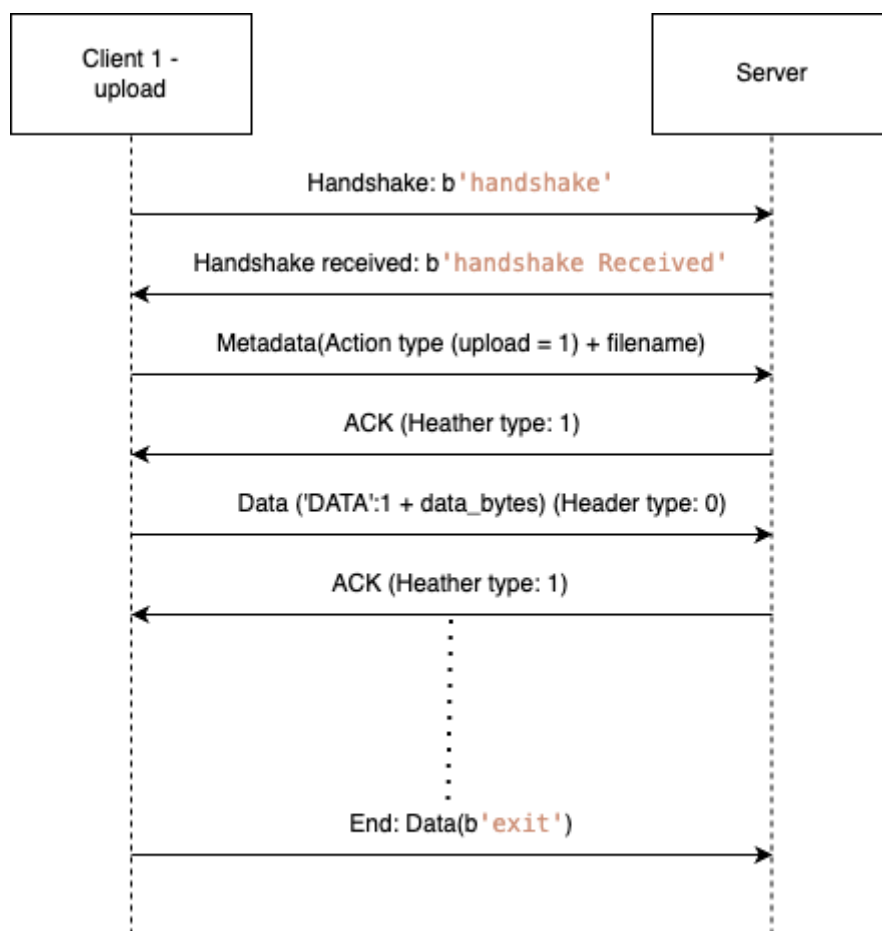


Fig. 2 Diagrama de secuencia del upload

Nótese que para cada cliente con el que se establezca una conexión, en el servidor se creará un thread para dicho cliente.

Cuando el cliente intente hacer un upload y el servidor no responda, se realizará un retry de dicho upload, hasta llegar a cinco intentos.

En caso de que al finalizar el upload y el cliente desee subir otro archivo, se le consultará por consola y podrá hacerlo de así desearlo.

Para el caso del download, el diagrama es análogo al anterior, con la salvedad de que la data correspondiente al archivo se envía desde el servidor, por ende no incluimos el diagrama correspondiente. A su vez, también se le consulta al cliente si desea descargar otro archivo al finalizar una descarga.

4. La capa de transporte del stack TCP/IP ofrece dos protocolos: TCP y UDP. ¿Qué servicios proveen dichos protocolos? ¿Cuáles son sus características? ¿Cuándo es apropiado utilizar cada uno?

a. TCP

Servicios que provee TCP:

- Confiabilidad: garantiza que los datos sean entregados en orden, sin pérdidas ni tampoco duplicaciones.
- Control de flujo: evita sobrecargar de datos a los receivers.
- Control de congestión: reduce la tasa de transmisión cuando detecta que hay congestión en la red.
- Orientación a la conexión: establece una conexión entre dos dispositivos antes de transmitir datos entre ellos.
- Verificación errores: valida que los datos no estén corruptos.
- Comunicación en ambos sentidos: permite la comunicación bidireccional
- Segmentación de datos: para tener una mejor transmisión de datos, segmenta los paquetes en pedazos más pequeños para facilitar el envío de estos.

Entonces, se debe usar TCP cuando la transmisión de datos debe ser confiable y también ordenada.

Por ejemplo, en los siguientes casos:

- Transferencia de archivos
- E-Mails
- Navegación web

b. UDP

Servicios que provee UDP:

- Sin conexión: Se envían los paquetes de datos sin confirmaciones de recepción ni tampoco retransmisiones.
- Rapidez: envía los paquetes sin confirmar su recepción. No mantiene una conexión ni tampoco la establece.
- Flexibilidad en el tamaño de paquetes: posibilidad de enviar datos de tamaños diversos
- Difusión: sirve para la transmisión de datos a más de un destinatario. Por ejemplo, streaming de video o audio.

Entonces, se debe usar UDP cuando la transmisión de datos debe ser rápida y eficiente.

Por ejemplo, en los siguientes casos:

- Streaming de video
- Online games
- Difusión/multidifusión unidireccional a muchos usuarios

Dificultades encontradas

En cuanto al trabajo en equipo, resultó dificultosa la división de tareas y paralelismo de las mismas. Dada la naturaleza de la aplicación, varios features requerían de ciertos módulos en común, lo cual conllevó a que diferentes miembros del equipo desarrollaran distintas implementaciones de ciertas funcionalidades (como por ejemplo el manejo de archivos). Otro de los desafíos que enfrentamos fue la comunicación. A pesar de nuestros esfuerzos por mantener una comunicación efectiva esto resultó en malentendidos y, en algunos casos, duplicación de esfuerzos.

En cuanto al protocolo en sí, enfrentamos un rendimiento deficiente en nuestro protocolo cuando se presentaba una pérdida de paquetes significativa en la red. Sin embargo, para abordar este problema, nos enfocamos en realizar ajustes clave en los tiempos de espera (timeouts) y el tamaño de los mensajes. A través de estos ajustes, logramos mejorar de manera sustancial la velocidad y la eficiencia de nuestra transferencia de archivos. Estos cambios permitieron que nuestro protocolo fuera más resistente a las condiciones adversas de la red, garantizando una experiencia más fluida y eficaz para los usuarios finales.

Conclusión

Durante el desarrollo de este trabajo práctico, nos enfrentamos al desafiante objetivo de crear una aplicación de red diseñada para la transferencia de archivos entre un cliente y un servidor. A lo largo de este proceso, exploramos minuciosamente las funciones de los protocolos de la capa de aplicación y, en particular, pusimos en práctica un protocolo basado en UDP para la transferencia de datos.

Esta experiencia nos proporcionó un conocimiento más profundo sobre los sistemas distribuidos y los protocolos que operan en las capas de aplicación y transporte. A pesar de los desafíos que surgieron en el camino, estamos seguros de que hemos alcanzado nuestros objetivos y hemos adquirido valiosa experiencia que indudablemente nos será de gran utilidad en futuros proyectos.

Bibliografía

- Computer Networking - A Top Down Approach, 7th - Jim Kurose.