(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
Nicolás Torres Dalmas	98439	ntorresdalma@fi.uba.ar

Índice

Introducción

El presente informe tiene como objetivo describir cómo fué el proceso de creación de la aplicación de red, se presentan los detalles del proyecto en el contexto del aprendizaje de sistemas distribuidos y protocolos de capa de aplicación y transporte.

Hipótesis y suposiciones realizadas

- Los hosts (clientes y servidor) cuentan con espacio suficiente para almacenar los archivos enviados.
- Se pueden utilizar las estrategias de stop and wait o selective repeat, pero teniendo en cuenta que la estrategia en todos los clientes debe ser la misma.
- Si un cliente se encuentra subiendo un archivo, otro cliente no podrá descargarlo hasta que el mismo se haya subido en su totalidad, ya que se perderán paquetes y se llegará timeout en otro caso.
- En caso de realizar un upload de un archivo con un nombre ya existente, se agregara al mismo archivo.

Implementación

Mininet

Para la realización de la topología, se crearon dos scripts que, mediante un único comando, permite crear y ejecutar en Linux la topología deseada.

Donde el primer parámetro es la cantidad de hosts (incluyendo al servidor) y el segundo el porcentaje de pérdida de paquetes.

Se decidió limitar la cantidad de hosts a diez, dado que si se crean muchas instancias de hosts es posible que la carga en términos de recursos como CPU y memoria se vuelva significativa. Cada host en Mininet se ejecuta como una instancia separada de un proceso de máquina virtual, dependiendo de la configuración. Por ende, si se crean muchos hosts, podría haber una sobrecarga en términos de administración de recursos. En resumen, al hacer pruebas utilizando una notebook con 4gb de ram, ya no funcionaba correctamente con más 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.

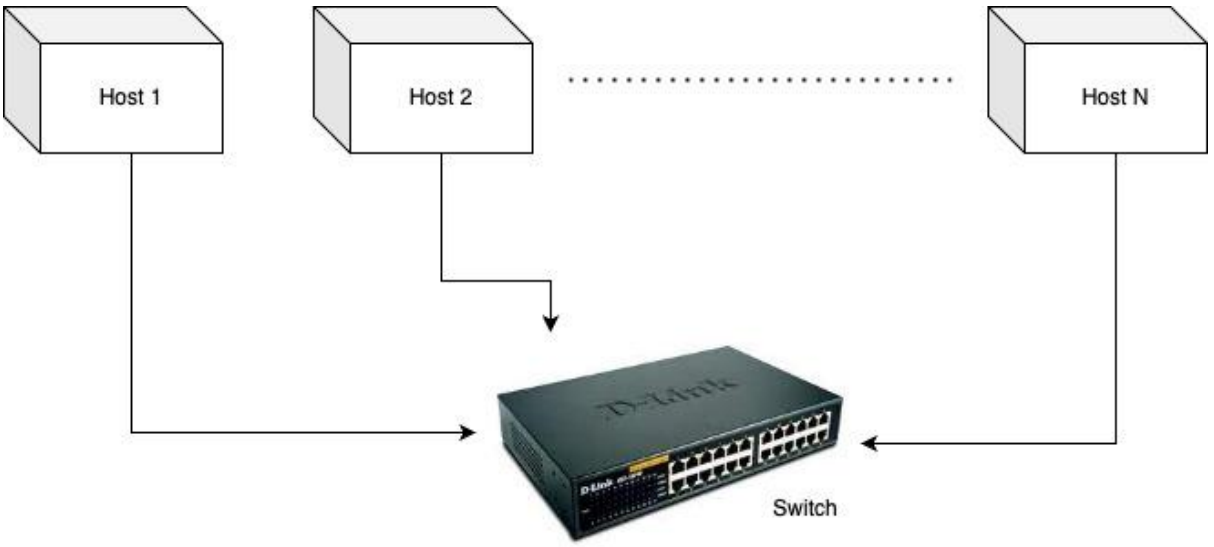


Fig. 1 Topologia de mininet

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

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

A continuación, dejamos 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):

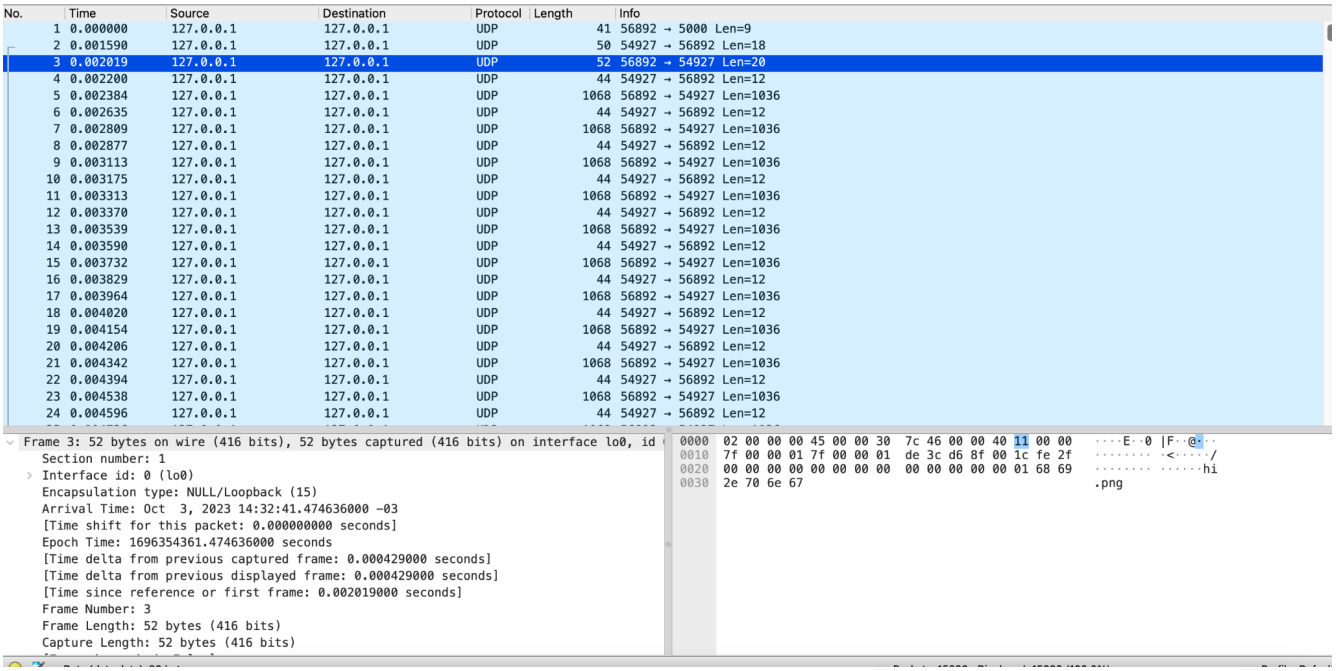
```
[→  src git:(master) ✗ python3 upload.py -n hi.png
chunk_size: 1023
INFO     Client upload started
INFO     Connected to server
file_name_bytes:  b'\x00\x01hi.png'
SENDING 0
ACK RECEIVED 0
SENDING 1
ACK RECEIVED 1
SENDING 2
ACK RECEIVED 2
SENDING 3
ACK RECEIVED 3
SENDING 4
```

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.

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



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
```

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:

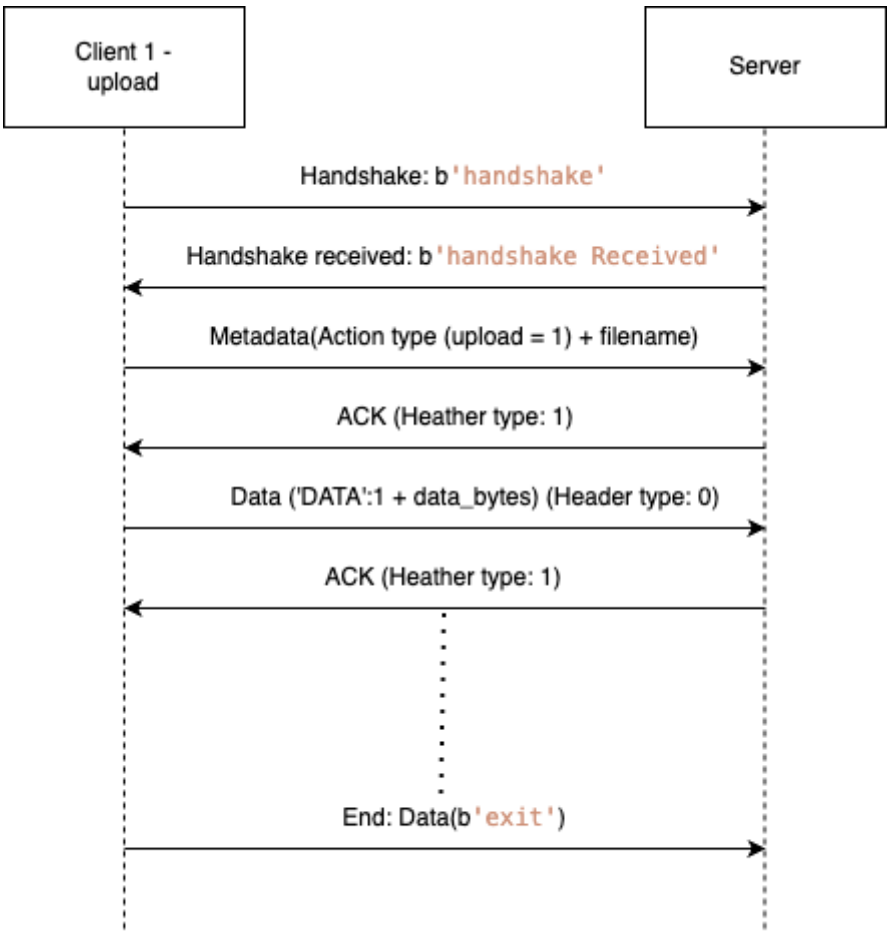


Fig. 2 Diagrama de secuencia del upload

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

Para el caso del download, 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.

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. Dada la naturaleza de la aplicación, varios features requerían de ciertos módulos en común, lo cual conllevo 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.

Conclusión

Durante el desarrollo de este trabajo práctico, nos enfrentamos al reto de crear una aplicación de red destinada a la transferencia de archivos entre un cliente y un servidor. Exploramos las funciones de los protocolos de capa de aplicación y pusimos en práctica un protocolo basado en UDP para la transferencia de datos.

Esta experiencia nos dió un entendimiento más profundo sobre los sistemas distribuidos y los protocolos de capa de aplicación y transporte. A pesar de los desafíos encontrados en el camino, creemos que hemos alcanzado nuestros objetivos y adquirido experiencia que nos será de utilidad en futuros proyectos.