Universidad de Buenos Aires

Facultad de Ingeniería

Introducción a los Sistemas Distribuidos (75.43)

Primer cuatrimestre de 2024

TP N ∘ 1: File Transfer - Grupo 11

Lenguaje: Python

Chávez Cabanillas, José	96467	jchavez@fi.uba.ar
Davico, Mauricio	106171	mdavico@fi.uba.ar
Elias, Federico	96105	felias@fi.uba.ar
Hoszowski, Juan	104557	jhoszowski@fi.uba.ar
Hoszowski, Luciana	104554	lhoszowski@fi.uba.ar

Índice

1. Introducción

En este trabajo práctico se implementa un File Transfer haciendo uso del protocolo de transporte UDP (User Datagram Protocol) con ciertas extensiones para que cumpla con las condiciones de un protocolo RDT (Reliable Data Transfer).

Desarrollamos una aplicación de arquitectura cliente-servidor que con las siguientes funciones:

- UPLOAD: Transferencia de un archivo del cliente hacia el servidor.
- DOWNLOAD: Transferencia de un archivo del servidor hacia el cliente.

Además, posee dos versiones: una con Selective Repeat y otra con Stop and Wait.

2. Hipótesis y suposiciones realizadas

1. Los paquetes enviados y recibidos nunca pueden estar corruptos
2. El cliente debe especificar el protocolo que desea utilizar para realizar la descarga o subida de un archivo.
3. Se asume que el tamaño del paquete son 4 KB.

3. Implementación

3.1. Paquete

SYN 1 B	ACK 1 B	Action 1 B	Protocol 1 B	Fin 1 B	Error 1 B	Sequence Number 4 B	Length 2 B	Payload 4084 B

- SYN: Utilizado en el Handshake para establecer conexión entre cliente-servidor
- ACK: Indica que se esta confirmando la llegada de un paquete
- Action: Utilizado por el cliente para indicar si se quiere descargar (0) o subir un archivo (1)
- Protocol: Utilizado por el cliente para indicar si se debe usar Stop and Wait (0) o Selective Repeat (1)
- Fin: Indica que se quiere finalizar la conexión
- Error: Utilizado por el servidor para indicar que el archivo solicitado no existe
- Length: Indica la longitud del payload (data del archivo que se envía)

3.2. Handshake

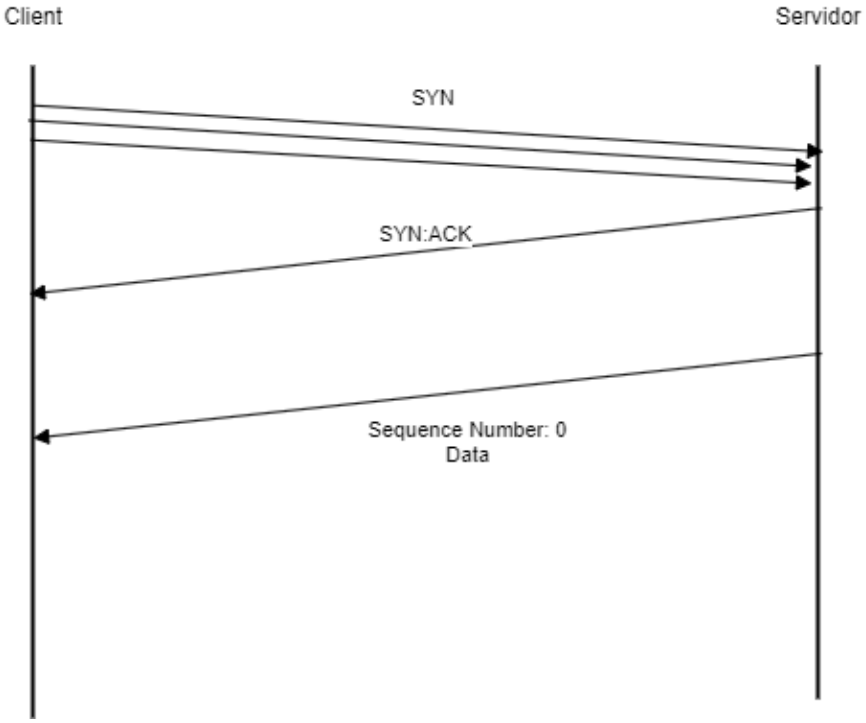


Figura 1: Handshake al hacer una descarga

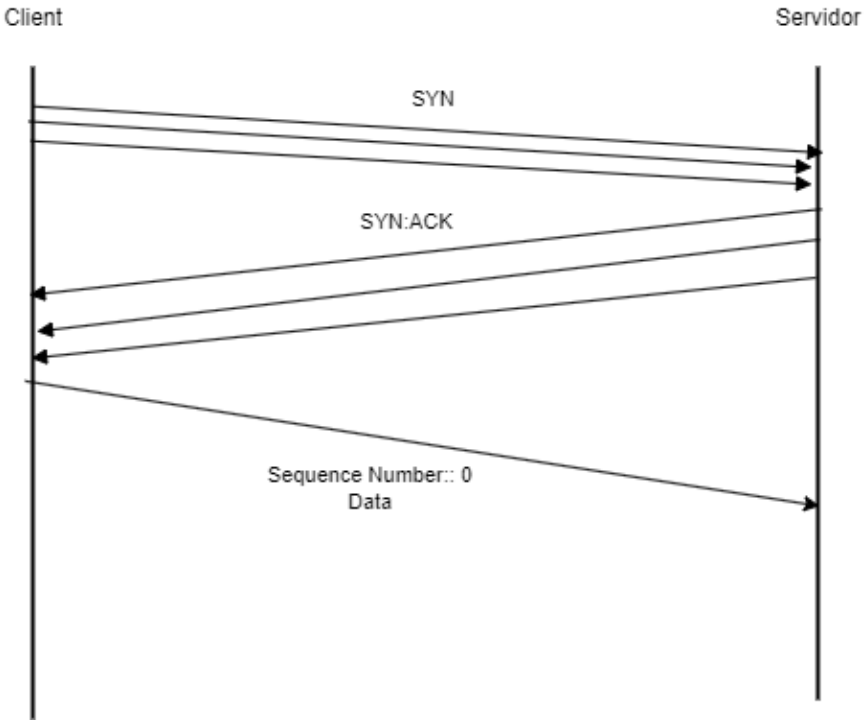


Figura 2: Handshake al hacer una subida

3.3. Protocolos

3.3.1. Stop and Wait

Cuando se envía un paquete se inicia un timer y se espera a recibir confirmación de que el otro host lo recibió para enviar el siguiente paquete, y si no se recibe respuesta una vez acabado el timer (TIMEOUT) se vuelve a enviar el paquete.

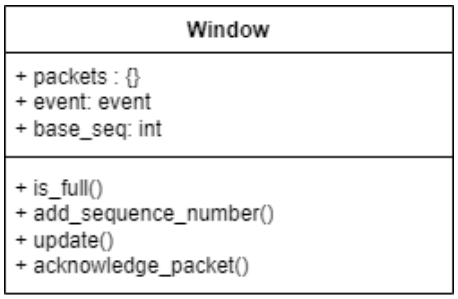
Uno de los problemas que trae este protocol es que al recibidor le pueden llegar paquetes duplicados, por lo que a cada paquete se le asigna un número de secuencia para permitirle identificar paquetes duplicados y descartarlos.

Cuando uno recibe un paquete, se responde prendiendo el flag ACK y enviando también el número de secuencia del paquete recibido en el campo *sequence number*.

3.3.2. Selective Repeat

Este protocolo permite enviar múltiples paquetes sin esperar un ACK, siempre que la cantidad de paquetes sin confirmación sea menor o igual a *N*. Cada paquete al momento de ser enviado inicia su propio timer, y se reenvía si no recibe un ACK dentro del tiempo definido.

Para implementar este protocolo se utilizó una clase *window* utilizado al momento de enviar un archivo. A continuación se muestra un diagrama de clase con sus atributos y métodos más importantes:



Cada vez que se envía un paquete se lo agrega al diccionario *packets*, siendo la clave su número de secuencia y el valor un booleano indicando si el paquete fue recibido o no.

Cuando se confirma la llegada de un paquete, se llama al método *update* que eliminara del diccionario de paquetes aquellos que fueron recibidos siempre que su número de secuencia sea igual a *baseSeq*.

Del lado del recibidor se implementa una mecánica similar a la de *update* al momento de decidir si debe escribir o no la data recibida, para asegurarse de que está escribiendo en orden.

4. Pruebas

Para testear nuestro trabajo realizamos 2 métodos complementarios:

- Ejecución + hash: Para ejecutar ya sea el cliente de UPLOAD o DOWNLOAD, se debe iniciar el SERVIDOR en una terminal y alguno de los clientes en otra terminal de acuerdo a los siguientes criterios de ejecución:

```
# start-server en una terminal
```

```
python3 start-server [-v|-q] -H <ip> -p <port> -s <path-directory>

# Upload server - Protocol Stop and Wait
python3 upload.py [-v|-q] -sw -H <ip> -p <port> -s <path-file>
-n <name-file>

# Upload server - Protocol Select Repeat
python3 upload.py [-v|-q] -sr -H <ip> -p <port> -s <path-file>
-n <name-file>

# Download server - Protocol Stop and Wait
python3 download.py [-v|-q] -sw -H <ip> -p <port> -d <path-file>
-n <name-file>

# Download server - Protocol Select Repeat
python3 download.py [-v|-q] -sr -H <ip> -p <port> -d <path-file>
-n <name-file>
```

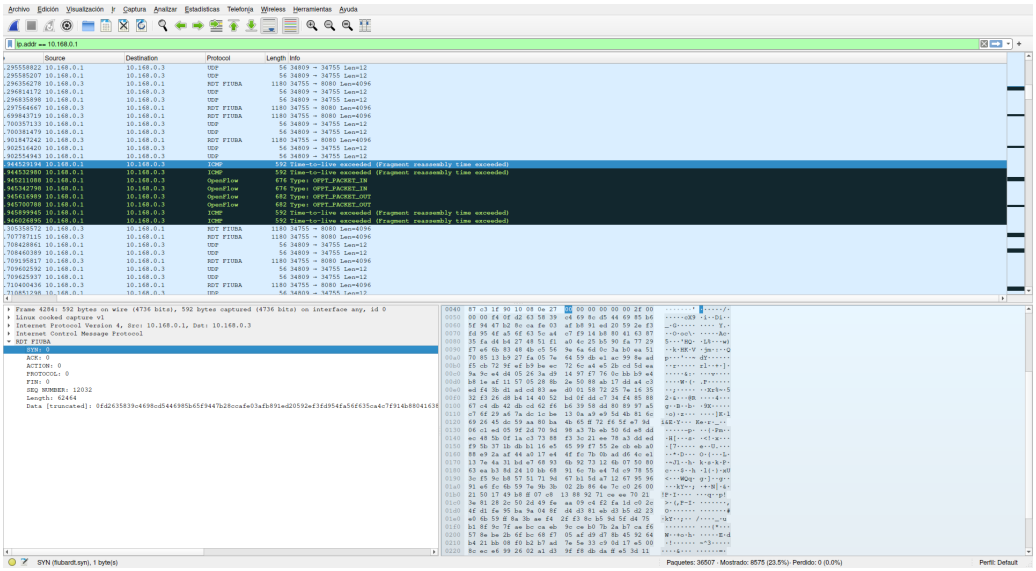
Para verificar que el archivo descargado o subido al servidor es igual al original, procedemos a calcular el hash mediante md5sum y verificar que ambos resultados sean iguales:

```
md5sum <path-file-original> <path-file-duplicated>
```

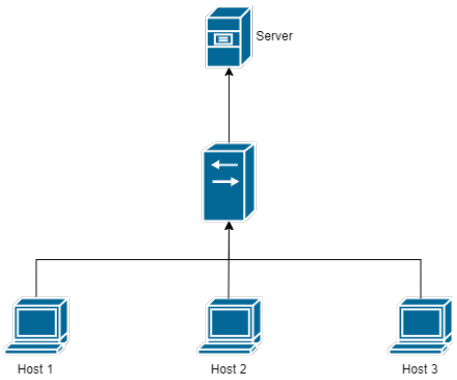
- Ejecución + Wireshark: Para constatar que los mensajes de nuestro protocolo son efectivamente como detallamos en este informe, necesitamos capturar paquetes con la herramienta Wireshark. Para esto, creamos un plugin que parsea los bytes de los mensajes de la capa de aplicación a los campos de nuestro protocolo. El mismo se encuentra en el repositorio con el nombre dissector.lua. Para utilizarlo, hay que seguir estos pasos:

  1. Tener instalado Wireshark
  2. Ir a Ayuda > Acerca de Wireshark > Carpetas y hacer click en la Ubicación del ítem Complementos Globales de Lua. Darle aceptar. En caso de no tener la carpeta creada, la creará.
  3. Copiar y pegar el dissector en esa carpeta y ya lo utilizará en las próximas capturas.

  Como se aprecia en la imagen siguiente, se muestra el parseo de los campos de nuestro protocolo, así como también, algunas pérdidas de paquetes en las pruebas realizadas. Con esto podemos dar un vistazo general a una de las tantas pruebas realizadas para el presente trabajo práctico.



También se debe hacer referencia que en cada una de las pruebas realizadas se elaboró una Topología hecha en Mininet, la cual tiene el siguiente modelo.



Para el funcionamiento de Mininet, primero que nada se debe instalar para su funcionamiento, para luego mediante un script en Python se pueda ejecutar mediante el siguiente comando en la terminal la topología. El script se encuentra en el repositorio del proyecto con el nombre mininet_topo.py.

```
# Mininet
sudo python3 mininet_topo.py
```

4.1. Mediciones

A continuación se adjunta un análisis comparativo de la transferencia de datos usando Stop And Wait o Selective Repeat con y sin packet loss, medida en segundos. Se testeó usando únicamente el cliente UPLOAD, para lo cual se hicieron 10 repeticiones por cada uno de los 2 archivos usados para las mediciones y se calculó el promedio del tiempo que se tardo en enviarse el archivo al SERVIDOR.

- Archivo 1: **foto.png**, 9.7mb
- Archivo 2: **data.jsonl**, 10mb

Stop And Wait

	9.7 MB	10 MB
0 %	0.917s	0,664s
10 %	247.619s	259,445s

Selective Repeat

	9.7 MB	10 MB
0 %	2,504s	3.44s
10 %	40,220s	63.57s

5. Preguntas a responder

5.1. Describa la arquitectura Cliente-Servidor.

La arquitectura cliente-servidor es un modelo de diseño para sistemas de computación distribuida en el que las responsabilidades y las tareas están divididas entre dos tipos de entidades: los clientes y los servidores.

El **cliente** es una aplicación o dispositivo que solicita servicios o recursos a través de una red. Puede ser una computadora, un smartphone, una tableta u otro dispositivo conectado a la red. Los clientes suelen ser usuarios finales que interactúan con la aplicación para realizar alguna tarea específica. Envían solicitudes al servidor y reciben respuestas.

Por otro lado, el **servidor** es una computadora o sistema especializado que proporciona servicios, recursos o datos solicitados por los clientes. Los servidores están diseñados para escuchar solicitudes entrantes, procesarlas y responder adecuadamente. Pueden ofrecer una amplia variedad de servicios, como almacenamiento de archivos, bases de datos, aplicaciones web, correo electrónico, etc. Los servidores suelen ser sistemas más potentes y especializados en tareas específicas como servidores web, servidores de base de datos, servidores de correo electrónico, entre otros.

En la arquitectura Cliente-Servidor, los clientes y servidores interactúan entre sí a través de una red, como Internet o una red local. La comunicación se realiza típicamente a través de un protocolo específico, como HTTP para aplicaciones web o TCP/IP para comunicaciones más genéricas.

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

La función de un protocolo de capa de aplicación es proporcionar servicios de comunicación específicos para las aplicaciones que se ejecutan en diferentes dispositivos. Estos protocolos operan en la capa más alta del modelo de referencia TCP/IP, la capa de aplicación, y son responsables de la comunicación de extremo a extremo entre las aplicaciones.

Los protocolos de capa de aplicación permiten a las aplicaciones interactuar entre sí a través de una red, definiendo cómo se formatean, transmiten y reciben los datos. Estos protocolos proporcionan una interfaz estándar y común para las aplicaciones, lo que facilita la comunicación entre diferentes sistemas y dispositivos.

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

El protocolo desarrollado en este trabajo está explicado en detalle en la sección 3 del presente informe.

5.4. 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?

TCP (Transmission Control Protocol) y UDP (User Datagram Protocol) son los dos protocolos principales en la capa de transporte del modelo TCP/IP. Proporcionan diferentes servicios y tienen características distintas que los hacen apropiados para diferentes situaciones:

TCP (Protocolo de Control de Transmisión):

- Servicios: TCP proporciona un servicio de entrega de datos confiable, orientado a la conexión. Se encarga de dividir los datos en segmentos, asegurar que los segmentos se entreguen en orden y retransmitir los segmentos perdidos o dañados. Además, ofrece control de flujo y control de congestión para garantizar que la red no se sobrecargue.
- Características:
  - Orientado a la conexión: antes de enviar datos, se establece una conexión entre el cliente y el servidor.
  - Entrega confiable: TCP garantiza que los datos lleguen correctamente y en orden.
  - Control de flujo y congestión: TCP ajusta la velocidad de transmisión para evitar la congestión de la red.
  - Requiere más recursos de red y procesamiento debido a la gestión de la conexión y la confiabilidad.
- Cuándo usarlo: Se utiliza cuando se necesita una entrega confiable de datos, como en aplicaciones web, transferencia de archivos, correo electrónico, transacciones bancarias en línea, entre otras.

UDP (Protocolo de Datagrama de Usuario):

- Servicios: UDP proporciona un servicio de entrega de datos no confiable y sin conexión. Simplemente envía los datagramas (paquetes) de forma independiente sin garantía de entrega o de orden de llegada.
- Características:
  - Sin conexión: no se establece una conexión antes de enviar datos.
  - Entrega no confiable: UDP no garantiza que los datos lleguen correctamente o en orden.
  - Menor sobrecarga: al no tener que gestionar la conexión ni la confiabilidad, UDP es más rápido y consume menos recursos que TCP.
- Cuándo usarlo: Se utiliza cuando la velocidad y la eficiencia son más importantes que la fiabilidad, como en transmisiones en tiempo real (voz sobre IP, videoconferencia, streaming de medios), juegos en línea, DNS (Domain Name System) y aplicaciones donde la pérdida ocasional de datos no es crítica.

6. Dificultades encontradas

Algunas de las dificultades encontradas a lo largo del proyecto, fueron los siguientes:

1. En algunos casos surgieron conflictos con las distintas versiones de Python.
2. Dificultades para instalar correctamente Mininet y que sea compatible con la versión de Python.
3. Encontrar el tamaño adecuado de bytes para el correcto envío de los paquetes.

7. Conclusión

Al concluir este proyecto, se logró desarrollar un protocolo de aplicación para la transferencia de archivos utilizando el protocolo de transporte UDP, con extensiones para asegurar una comunicación estable y segura. Se implementaron dos versiones del protocolo, una basada en el esquema Stop and Wait y otra en el esquema Selective Repeat. A lo largo de esta tarea se abordaron diversos desafíos, desde la definición precisa de los mensajes hasta la gestión eficiente de los tiempos de espera y la prevención de la pérdida de paquetes, entre otros aspectos cruciales de la interacción cliente-servidor.

El resultado final es un programa funcional que permite a los usuarios cargar y descargar archivos de manera fiable utilizando una conexión UDP. Además, se llevaron a cabo pruebas exhaustivas, incluyendo escenarios de pérdida de paquetes con archivos de diferentes tamaños, con el objetivo de validar la robustez y la confiabilidad del protocolo.

En resumen, este proyecto proporcionó una comprensión profunda de los desafíos asociados con la implementación de un protocolo de aplicación, al mismo tiempo que brindó una valiosa experiencia en el diseño y desarrollo de sistemas de comunicación confiables.