FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 1

[TA048] Redes
Primer cuatrimestre de 2025

Alumno	Padron	Email
AVALOS, Victoria	108434	vavalos@fi.uba.ar
CASTRO MARTINEZ, Jose Ignacio	106957	jcastrom@fi.uba.ar
CIPRIANO, Victor	106593	vcipriano@fi.uba.ar
DEALBERA, Pablo Andres	106858	pdealbera@fi.uba.ar
DIEM, Walter Gabriel	105618	wdiem@fi.uba.ar

Índice

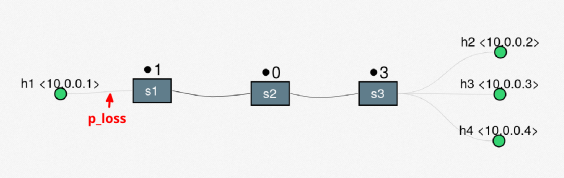
1. Introduccion

2. Hipótesis y suposiciones realizadas

- La carga/descarga no va a conservar la metadata del archivo. Es decir, si yo descargo un archivo, ese archivo va a tener metadata como si yo hubiera creado el archivo desde cero usando 'touch archivo'.
- Si el cliente utiliza otro protocolo para comunicarse con el server, el server debe rechazar este pedido. (PROTOCOL MISSMATCH). El header tendra un campo dedicado a esto.
- El argumento de FILENAME sera opcional, en caso de no estar, se utiliza el nombre original del archivo.
- Por simplicidad, vamos a guardar todos los archivos en DIRPATH sin ningun nivel de sub-directorios.
- Por simplicidad, vamos a tener un tamaño maximo de 2GB para la subida y descarga de archivos.
- Los archivos en proceso de escritura se van a escribir en una ubicacion temporal para evitar que se corrompan en la ubicacion que el cliente pidio.
- Usar seek y bufferear para leer el archivo. Leer con slices del buffer.

3. Implementacion

3.1. Topología



La topología es una red lineal con 1 host servidor conectado a 3 switches en serie cuyo ultimo switch esta conectado a n hosts clientes. El primer enlace (el conectado entre el servidor y el primero switch) tiene configurado un packet loss del 10\

3.2. Especificación del protocolo Stop-and-Wait

3.2.1. General

1. Tamaño maximo de payload

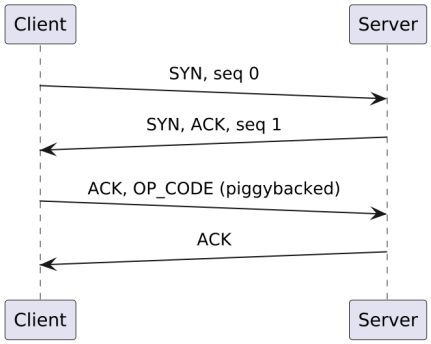
   El tamaño máximo de payload es el tamaño máximo de un datagrama UDP menos el tamaño de los headers de IP, UDP y SAW.

   Como el MTU que usamos en mininet es 1500, el tamaño máximo de payload es:

$$\begin{aligned} \text{HISTORICAL\_MTU} &= 1500 \\ \text{MAX\_IP\_HEADER\_SIZE} &= 60 \\ \text{UDP\_HEADER\_SIZE} &= 8 \\ \text{SAW\_PROTOCOL\_HEADER\_SIZE} &= 6 \end{aligned}$$

$$\begin{aligned}\text{FILE\_CHUNK\_SIZE} = \text{HISTORICAL\_MTU} - \text{MAX\_IP\_HEADER\_SIZE} \\ - \text{UDP\_HEADER\_SIZE} - \text{SAW\_PROTOCOL\_HEADER\_SIZE}\end{aligned}$$

3.2.2. Handshake



La idea es usar este handshake para inicialización de recursos del servidor y check de protocolo.

Se usa la nomenclatura S para mencionar al servidor y C para el cliente.
Mensajes para caso Download y caso Upload:

1. C → S: con flag SYN para declarar una solicitud de conexion y el protocolo

Flujo normal (mismo protocolo):

2. S → C: con flag de SYN y ACK para declarar que se acepta la conexión y el puerto donde se va a escuchar el resto.
3. C → S: con flag ACK al mismo welcoming socket.

Flujo de error (distinto protocolo):

2. S → C: con flag FIN para denegar la conexión por usar un protocolo distinto.

Se hace una transferencia de puerto para que el welcoming socket se encargue solamente de establecer conexiones y el nuevo puerto maneje la transferencia de datos del archivo. El último ACK de parte del cliente asegura que se recibió el puerto donde se tiene que comunicar y es seguro hacer el cambio de socket.

3.2.3. Etapa de configuración y Transferencia

El cliente ya sabe que tiene que comunicarse con el nuevo puerto.
Se envia primero la configuracion para saber si la operacion es valida y tener en cuenta casos de error, y luego se hace la transferencia.

Mensajes para caso Download y caso Upload:

1. C → S: se declara la operacion (OP), que puede ser download (0) o upload (1)
2. S → C: ACK de la operacion (no falla)

Continuación de mensajes para caso Download:

3. Mensaje 3 C → S: filename

Flujo Normal:

4. S → C: ACK + comienzo de datos (piggybacked)
5. C → S: ACK
6. S → C: continuacion de datos

Flujo de error (no existe un archivo con ese nombre):

4. S → C: FIN, se termina la conexión

Continuación de mensajes para caso Upload:

3. C → S: filename

Flujo de error (ya existe un archivo con ese nombre):

4. S → C: FIN, se termina la conexión

Flujo normal:

4. S → C: ACK
5. C → S: filesize

Flujo de error (archivo es más grande que el tamaño máximo o [TODO] no hay más espacio en disco):

6. S → C: FIN, se termina la conexión

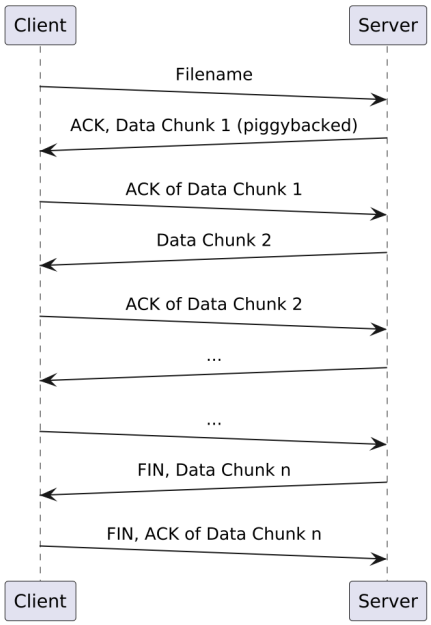
Flujo normal:

6. S → C: ACK
7. C → S: comienzo de datos
8. S → C: ACK
9. C → S: continuacion de datos

3.2.4. Cierre

El flag FIN va piggybacked con la última data para que sea más eficiente. El receptor confirma con un ACK + FIN para que el emisor sepa que le llego la informacion, y por si este se pierde está el último ACK para confirmar el cierre de parte del emisor.

1. Mensajes para caso Download

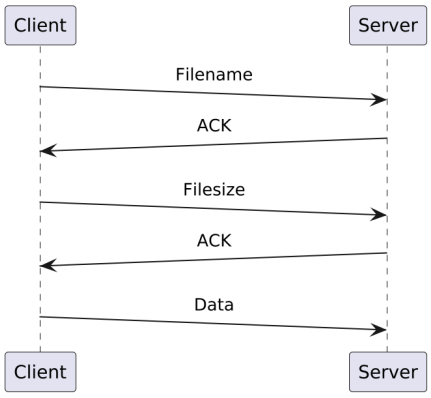


a) S → C: ultima data, va piggybacked el flag FIN

b) C → S: ACK + FIN

c) S → C: ACK

2. Mensajes para caso Upload:



a) C → S: ultima data, va piggybacked el flag FIN

b) S → C: ACK + FIN

c) C → S: ACK

4. Pruebas

4.1. Tabla de Datos de Wireshark

No	Time	Src	Dst	Proto	Len	Type	SEQ	ACK	SYN	FIN	SrcPort	DstPort
1	0.000000000	10.0.1.1	10.0.0.1	SAW	48	Stop-and-Wait	0	False	True	False	52515	0
2	0.000191297	10.0.0.1	10.0.1.1	SAW	48	Stop-and-Wait	0	True	True	False	41367	0
3	0.002208402	10.0.1.1	10.0.0.1	SAW	50	Stop-and-Wait	1	True	False	False	52515	2
4	0.002801150	10.0.0.1	10.0.1.1	SAW	48	Stop-and-Wait	1	True	False	False	41367	0
5	0.004355272	10.0.1.1	10.0.0.1	SAW	53	Stop-and-Wait	0	False	False	False	52515	5
6	0.004722710	10.0.0.1	10.0.1.1	SAW	48	Stop-and-Wait	0	True	False	False	41367	0
7	0.005754904	10.0.1.1	10.0.0.1	SAW	52	Stop-and-Wait	1	False	False	False	52515	4
8	0.005879502	10.0.0.1	10.0.1.1	SAW	48	Stop-and-Wait	1	True	False	False	41367	0
9	0.006562696	10.0.1.1	10.0.0.1	SAW	1474	Stop-and-Wait	0	False	False	False	52515	1426
10	0.006634214	10.0.0.1	10.0.1.1	SAW	48	Stop-and-Wait	0	True	False	False	41367	0
11	0.006819155	10.0.1.1	10.0.0.1	SAW	1474	Stop-and-Wait	1	False	False	False	52515	1426
12	0.006887880	10.0.0.1	10.0.1.1	SAW	48	Stop-and-Wait	1	True	False	False	41367	0
...	...	...	...	...	...	...	...	...	...	...	...	.....
384	0.036747322	10.0.0.1	10.0.1.1	SAW	48	Stop-and-Wait	1	True	False	False	41367	0
385	0.036806828	10.0.1.1	10.0.0.1	SAW	1474	Stop-and-Wait	0	False	False	False	52515	1426
386	0.036860606	10.0.0.1	10.0.1.1	SAW	48	Stop-and-Wait	0	True	False	False	41367	0
387	0.037000220	10.0.1.1	10.0.0.1	SAW	1474	Stop-and-Wait	1	False	False	False	52515	1426
388	0.037084310	10.0.0.1	10.0.1.1	SAW	48	Stop-and-Wait	1	True	False	False	41367	0
389	0.037217987	10.0.1.1	10.0.0.1	SAW	363	Stop-and-Wait	0	False	False	True	52515	315
390	0.037459011	10.0.0.1	10.0.1.1	SAW	48	Stop-and-Wait	0	True	False	False	41367	0
391	0.037511183	10.0.0.1	10.0.1.1	SAW	48	Stop-and-Wait	0	False	False	True	41367	0
392	0.037580397	10.0.1.1	10.0.0.1	SAW	48	Stop-and-Wait	1	True	False	False	52515	0

4.2. Análisis de la comunicación Stop-and-Wait

La tabla presentada al inicio del informe muestra una traza de paquetes intercambiados entre dos nodos (10.0.1.1 y 10.0.0.1) utilizando un protocolo confiable de transferencia de datos basado en el esquema Stop-and-Wait. Este protocolo garantiza la entrega ordenada y libre de errores mediante el envío secuencial de paquetes, esperando una confirmación (ACK) por cada uno antes de continuar con el siguiente.

El ciclo de vida de la comunicación puede dividirse en tres fases:

1. Establecimiento de la conexión:

- El cliente (10.0.1.1) inicia la conexión enviando un paquete con la bandera 'SYN' activada.
- El servidor (10.0.0.1) responde con un paquete que contiene tanto 'SYN' como 'ACK', indicando aceptación.
- Finalmente, el cliente responde con un 'ACK', completando el procedimiento de handshake.

2. Transferencia de datos:

- Una vez establecida la conexión, el cliente comienza a enviar datos, alternando los números de secuencia (SEQ) entre 0 y 1. Este comportamiento es característico del protocolo Stop-and-Wait.
- Por cada paquete de datos enviado, el servidor responde con un paquete de confirmación ('ACK') para indicar que ha recibido correctamente el contenido.
- El campo 'Len' refleja el tamaño de los datos transportados, y los puertos de origen y destino se utilizan para mantener la sesión activa entre los procesos involucrados.

3. Finalización de la conexión:

- El cliente inicia el cierre de la sesión enviando un paquete con la bandera 'FIN' activada.
- El servidor responde primero con un 'ACK', y luego con su propio paquete 'FIN', indicando que también desea cerrar la conexión.
- Finalmente, el cliente responde con un 'ACK', completando el cierre de la comunicación de manera ordenada.

En resumen, esta captura de paquetes evidencia el funcionamiento correcto de una implementación del protocolo Stop-and-Wait, en donde cada paquete enviado es seguido por una respuesta de confirmación, y el inicio y cierre de la conexión se realizan mediante un protocolo de control fiable. Este análisis resulta fundamental para comprender el modelo de servicio que la capa de transporte ofrece a las aplicaciones.

5. Preguntas a Responder

5.1. Describa la arquitectura Cliente-Servidor.

La arquitectura Cliente-Servidor es una de dos arquitecturas más comunes. En esta arquitectura hay un *host* (*end system*) llamado *server* que esta siempre encendido que pasivamente escucha *requests* de otros *hosts* llamados *clients* que son agentes activos que inician la comunicación con el *server*.

Un ejemplo de esta arquitectura es una aplicacion Web donde hay un *Web server* que escucha *requests* de navegadores web. El navegador web es el cliente que inicia la comunicación y el *Web server* es el servidor que responde a los *requests*. Estos mensajes tienen el formato de Capa de Aplicación HTTP.

5.1.1. Caracteristicas

- Los clientes son agentes activos que inician la comunicación.
- Los clientes no se comunican entre si.
- Los clientes no necesitan estar encendidos todo el tiempo ni tener una IP fija.
- Los servidores son pasivos y siempre están encendidos.
- Los servidores **deben** tener una IP fija bien conocida (*well-known IP address*) que se puede resolver con un nombre de dominio DNS (*domain name*).
- Los servidores pueden tener múltiples clientes conectados al mismo tiempo.

5.1.2. Ventajas

- Diseño simple usando protocolos sin estado como HTTP donde el servidor no necesita mantener informacion sobre clientes ya que se puede guardar informacion del cliente en *cookies* del cliente y estos se transmitidos en *headers* HTTP.
- Puede soportar un gran numero de clientes.

5.1.3. Desventajas

- Un solo punto de falla. Si el servidor se cae, el servicio se cae.
- El servidor debe estar encendido todo el tiempo.
- Gran costo para escalar, ya que a medida de que el servicio tiene mas usuarios, el servidor debe tambien aumentar su capacidad de procesar mas clientes.

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

Un protocolo de capa de aplicación especifica cómo los procesos de una aplicación, que se ejecutan en diferentes sistemas finales, intercambian mensajes entre sí. Este tipo de protocolo define:

- Los tipos de mensajes que se envían, como mensajes de solicitud y de respuesta.
- La sintaxis de los mensajes, es decir, la estructura de los campos dentro de cada mensaje y cómo se separan o identifican esos campos.
- La semántica de los campos, indicando qué significa la información contenida en cada uno.
- Las reglas de comunicación, que establecen cuándo un proceso debe enviar un mensaje y cómo debe reaccionar al recibir uno.

En resumen, los protocolos de capa de aplicación aseguran que las aplicaciones puedan comunicarse correctamente y coordinarse en la red, haciendo posible servicios como el correo electrónico, la web o la transferencia de archivos.

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

5.4. La capa de transporte del stack TCP/IP ofrece dos protocolos: TCP y UDP.

5.4.1. ¿Qué servicios proveen dichos protocolos?

Ambos protocolos proveen los siguientes servicios:

- **Multiplexación/Demultiplexación**: son los mecanismos que permiten extender el servicio de entrega de IP entre dos end systems a un servicio de entrega entre dos procesos que se ejecutan en esos sistemas. Dichos mecanismos permiten identificar a qué proceso pertenece cada ssegmento recibido.
- **Chequeo de integridad**: se verifica que no haya errores en los datos mediante un campo de checksum en los headers de ambos protocolos.

UDP no realiza ninguna otra función extra. Por lo tanto, su servicio es:

- **No confiable**: no garantiza que la entrega de los paquetes sea exitosa, ni tampoco que lleguen en orden.
- **Sin conexión**: cada paquete datagrama se envía de manera independiente, sin garantías de que el receptor esté listo o incluso disponible.

Por su parte, TCP ofrece las siguientes funcionalidades adicionales:

- **Orientado a la conexión**: antes de que un proceso de aplicación pueda comenzar a enviar datos a otro, ambos procesos deben comunicarse entre sí; es decir, deben enviarse algunos segmentos preliminares para establecer los parámetros de la transferencia de datos subsiguiente. Se trata de una conexión lógica con un estado en común que reside en TCP de los hosts.
- **Transferencia de datos confiable**: garantiza la entrega, el orden y la no corrupción de los datos. Esto lo logra mediante timers, numeros de secuencia y ACKs (flags que indican que un paquete fue entregado correctamente).
- **Control de congestión**: festiona que no se saturen los enlaces. Es más bien un servicio para la red.
- **Control de flujo**: para eliminar la posibilidad de que el remitente desborde el búfer del receptor. Hace coincidir la velocidad a la que el remitente envía con la velocidad a la que la aplicación receptora lee.

5.4.2. ¿Cuáles son sus características?

Algunas de las características de UDP son las siguientes:

- **Pequeño overhead de header por paquete**: UDP posee un header pequeño (8 bytes) en comparación con TCP (20 bytes)
- **Sin estado de conexión**: UDP no mantiene un estado de conexión en los end systems, por lo que no rastrea ningún parámetro. Por esta razón, un servidor dedicado a una aplicación específica generalmente puede admitir muchos más clientes activos cuando la aplicación se ejecuta mediante UDP en lugar de TCP.
- **Sin retraso por conexión**: UDP no induce ningún retraso para establecer una conexión, a diferencia de TCP que posee un handshake de tres pasos.

Por su parte, TCP posee las siguientes características:

- **Full-duplex**: dada una conexión TCP entre dos hosts, digamos A y B, la información puede fluir de A a B al mismo tiempo que fluye información de B a A.
- **Conexión point-to-point**: la conexión de TCP únicamente se puede establecer entre un único remitente y un único receptor, no admite multicasting.
- **Three-Way Handshake**: para establecer la conexión mencionada anteriormente se realiza un procedimiento donde se envían tres segmentos.

5.4.3. ¿Cuando es apropiado utilizar cada uno?

Ninguno de estos protocolos es mejor que el otro. Para decidir cuál de ellos utilizar, se deben tener en cuenta las necesidades de la aplicación. Debido a las características mencionadas anteriormente, UDP resulta más apropiado para aplicaciones que requieran mayor velocidad sin que sea tan sensible a algunas pérdidas de paquetes, por ejemplo plataformas de streaming, y si se tiene un servidor dedicado a una aplicación específica que necesita poder admitir muchos más clientes activos. Por otro lado, TCP es más ventajoso para las aplicaciones que necesitan un transporte confiable de los datos. Algunos ejemplos son el email y la web.

6. Dificultades Encontradas

7. Conclusion

8. Anexo: Fragmentacion IPv4

8.1. Analisis