Universidad de Buenos Aires

Facultad de Ingeniería

1er Cuatrimestre de 2022

[75.43]

Introducción a los sistemas distruibuídos

Trabajo práctico: File Transfer

Integrantes: *Padrón:*

Fusco, Matias <mfusco@fi.uba.ar> 105683
Sardella, Florencia <fsardella@fi.uba.ar> 105717
Semorile, Gabriel <gsemorile@fi.uba.ar> 105681
Su, Agustina Doly <asu@fi.uba.ar> 105708
Szwarcberg, Tomás <tszwarcberg@fi.uba.ar> 103755

Links:

- Repositorio - GitHub.

Índice

parano

1. Introducción

El presente trabajo práctico tiene como objetivo la creación de una aplicación de red. Para tal finalidad, será necesario comprender cómo se comunican los procesos a través de la red, y cuál es el modelo de servicio que la capa de transporte le ofrece a la capa de aplicación. Además, para poder lograr el objetivo planteado, se aprenderá el uso de la interfaz de sockets y los principios básicos de la transferencia de datos confiable.

Se plantea como objetivo la comprensión y la puesta en práctica de los conceptos y herramientas necesarias para la implementación de un protocolo RDT. Para lograr este objetivo, se deberá desarrollar una aplicación de arquitectura cliente-servidor que implemente la funcionalidad de transferencia de archivos mediante las siguientes operaciones:

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

Dada las diferentes operaciones que pueden realizarse entre el cliente y el servidor, se requiere del diseño e implementación de un protocolo de aplicación básico que especifique los mensajes intercambiados entre los distintos procesos.

2. Hipótesis y suposiciones realizadas

A continuación se listan las hipótesis y suposiciones que se han tenido en cuenta para la realización de este trabajo práctico:

- No llegan paquetes corruptos a la capa de aplicación
- Si el cliente quiere subir un archivo al servidor con un nombre ya existente, se reemplaza su contenido
- Si el cliente quiere descargar un archivo del servidor con un nombre ya existente en su dominio se reemplaza su contenido.
- Si el host dado en el cliente no es válido o no está aceptando conexiones en el puerto indicado, el cliente finaliza su ejecución
- El tamaño máximo de un archivo a transferir debe ser 4GiB
- La ruta a un archivo puede tener como máximo 256 caracteres

3. Implementación

3.1. Servidor

El servidor es el encargado de recibir mensajes de distintos clientes de manera concurrente. Para ello, el servidor debe estar corriendo antes de que los clientes quieran mandar alguna request. Con el objetivo de atender a los distintos clientes concurrentemente se diseñó una secuencia de threads. El hilo principal es el main (lee input del usuario), desde el mismo se inicializa el thread acceptor, quien espera a recibir conexiones en el caso de TCP o mensajes en el caso de UDP. Una vez que el acceptor recibe una conexión o llega una request de algún cliente se lanza un hilo específico por cada uno de ellos. En el caso de UDP, se crea un hilo específico para la transferencia de mensajes con ese cliente, donde se encolan los distintos mensajes del cliente en una cola. Luego, el servidor irá leyendo de las distintas colas para atender las requests.

Con el objetivo de implementar distintos protocolos haciendo que todos garanticen RDT se crearon 3 clases para representar a los distintos servidores, estas son: Server, SAWServer y GBNServer. La clase Server funciona bajo el protocolo TCP, las clases SAWServer y GBNServer bajo UDP pero implementando Stop & wait y Go Back N respectivamente.

En el caso de TCP, la clase Server inicia el hilo aceptor mientras que el hilo principal espera a recibir una ”q”por parte del usuario utilizando la consola de comandos para cortar la ejecución del servidor. El hilo Acceptor espera a recibir conexiones y en caso de que una llegue, inicia un nuevo hilo destinado a esa conexión con el cliente. Este hilo específico es el TCPConnection, el cuál dependiendo del tipo de request que haga el cliente la maneja de distintas maneras. El cliente puede hacer un descarga o una subida de un archivo.

En el caso de UDP, las respectivas clases (SAWServer y GBNServer) inician el hilo UDPAcceptor. El mismo espera recibir mensajes de los clientes. Cuando recibe un mensaje chequea si ya existía una conexión con ese cliente. En caso de que no exista, inicia un hilo específico (UDPConnection) para la transferencia de mensajes y, en caso contrario, agrega a una cola específica por cliente el paquete recibido.

Para manejar las dos request de los clientes, se instancia la clase FileSender para descarga y la clase FileReceiver para subida, la primera hace validaciones sobre el archivo solicitado y luego envía un mensaje de confirmación al cliente, la segunda valida que el tamaño del archivo que quiere subir el cliente sea válido y que haya el suficiente espacio en memoria y luego envía un mensaje de confirmación.

El mensaje de confirmación para ambos casos se conforma de:

- número de secuencia: 4 bytes que representan el número de secuencia (en el caso de TCP este valor no se tiene en cuenta debido a que TCP ya implementa RDT por su cuenta, se utiliza este campo por compatibilidad entre los distintos protocolos y sus versiones).
- estado de operación: 1 byte con el estado de la operación, donde un 0 representa que la operación es posible y un 1 que la operación dió error.
- tamaño del archivo: 4 bytes representando el tamaño del archivo a enviar (en caso de descarga, completo de 0s en otro caso).

Con el objetivo de poder manejar la sincronización de los mensajes en los distintos protocolos y sus versiones se optó por designar un tamaño fijo para todos los paquetes que son envíados por la red, el mismo es de 4096 bytes.

En caso de descarga, el servidor lee el archivo teniendo en cuenta el tamaño fijo de los paquetes y envía los datos al cliente. Este mensaje se conforma de:

- número de secuencia: 4 bytes que representan el número de secuencia (en el caso de TCP este valor no se tiene en cuenta debido a que TCP ya implementa RDT por su cuenta, se utiliza este campo por compatibilidad entre los distintos protocolos y sus versiones).
- tamaño de payload: 4 bytes indicando el tamaño del payload neto envíado.
- payload/data: como máximo se podrán envíar 4088 bytes de payload debido a la restricción impuesta por el tamaño fijo de los paquetes.

En el caso de una subida, el servidor recibe los paquetes del cliente que respetan los mismos campos nombrados arriba.

En el caso de TCP, los números de secuencia no se tuvieron en consideración debido a que TCP ya garantiza RDT. En cambio, en el caso de UDP, se utilizaron estos campos de los paquetes para verificar que lleguen de un host al otro y para poder ordenarlos.

En nuestro diseño optamos por elegir que el campo del número de secuencia se use para:

- Indicar que número de paquete representa si es un paquete que transfiere datos del archivo.
- Indicar el número de secuencia del paquete que se quiere recibir en caso de que el mensaje funcione como un acknowledge.

En el caso de Stop & wait y de Go Back N, los ackowledges se usan para indicarle al otro host el número de paquete que uno espera.

En la implementación de stop & wait, cuando se envía un paquete se espera a que la recepción del mismo sea confirmada antes de mandar otro. En caso de que el cliente envie una confirmación con un número de secuencia menor al que esperaba el servidor, se vuelve a envíar el paquete, ya que esto indica que no le llegó al cliente.

En la implementación de Go Back N, se implementó una ventana de $N = 5$. La ventana se refiere a la cantidad de paquetes que se pueden envíar por vez sin confirmar. De esta manera, el servidor envía al cliente N paquetes y luego espera la confirmación de los mismos. Si los números de secuencia de las distintas confirmaciones no son los que esperabamos entonces se reenvían los N paquetes nuevamente. En cambio, si las confirmaciones son correctas entonces la ventana se desliza para tener en cuenta a los siguientes paquetes a enviar (se incrementa el número de secuencia).

3.2. Cliente

El cliente es quien inicia la conexión con el servidor, esto lo hace mandandole una request al puerto y la dirección correspondiente del servidor (en el caso de TCP antes se requiere establecer una conexión).

Este mensaje de request se compone de:

- Número de secuencia: 4 bytes que indican el número de secuencia del paquete (campo no utilizado en la implementación de TCP).
- Tipo de operación: 1 byte indicando si el tipo de operación es de tipo descarga (0) o de tipo subida (1).
- Largo del path: 1 byte indicando el largo del path del archivo.
- Path: 'Largo del path' bytes con el path del archivo.
- Tamaño del archivo: 4 bytes indicando el tamaño del archivo a enviar (en caso de descarga, en caso de subida se completa con ceros).

Como se nombró anteriormente en el servidor, los paquetes tienen un tamaño fijo de 4096 bytes por lo que si faltan bytes para completar el paquete se rellenan con ceros. Esto se hace con el objetivo de evitar el desfasaje de mensajes en el caso de la implementación de los protocolos UDP ya que al haber pérdida de paquetes se puede modificar el orden de la secuencia de mensajes (y así confundir un mensaje que tenía menos bytes con otro de más bytes o viceversa).

En caso de que el tipo de operación sea una subida entonces, el cliente leerá el archivo en cuestión y enviará paquetes al servidor con la data del mismo. La confirmación de este mensaje es idéntica a la nombrada en el servidor.

Con el objetivo de lograr una transferencia confiable de datos al utilizar el protocolo UDP, se implementaron también del lado del cliente dos versiones: una utilizando Stop wait y otra utilizando Go Back N.

Para lograr que garanticen RDT se utilizan los campos de número de secuencia y se envían acknowledges (confirmaciones). Los mensajes de confirmación se conforman de:

- Número de secuencia: 4 bytes que indican el número de secuencia del paquete esperado.

Al igual que con el resto de los paquetes el mismo se completa con ceros para llegar al tamaño fijo esperado.

Stop & wait y Go Back N funcionan análogamente a lo nombrado para el servidor.

En el caso de Stop & wait cuando se envía un paquete se espera una confirmación de recepción antes de envíar el próximo. Si el número de secuencia de la confirmación es el esperado entonces se manda un nuevo paquete con un número de secuencia mayor. En caso contrario, se envía nuevamente el paquete hasta que sea confirmado. Desde el punto de vista del host receptor, si el paquete recibido tiene el número de secuencia esperado entonces se manda un mensaje confirmando su llegada (se envía con un número de secuencia mayor) y se procesa el paquete, en caso contrario, se envía el número de secuencia que uno esperaba recibir y se droppea el paquete.

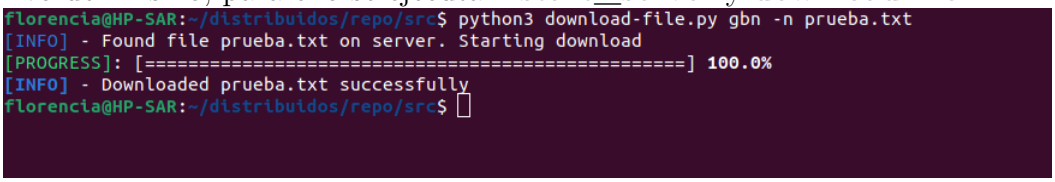
En el caso de Go Back N se envían N paquetes (N indica el tamaño de la ventana actual). Una vez envíados se espera la confirmación de los mismos para poder desplazar la ventana y envíar los paquetes siguientes.

4. Pruebas

Una vez implementados los distintos protocolos, se proseguió en probarlos para verlos en funcionamiento.

4.1. Descarga de un archivo

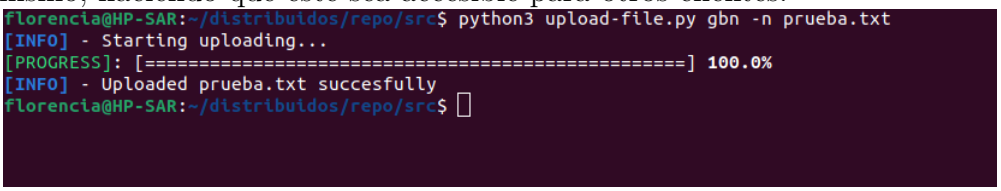
En este caso se prueba que el cliente pueda conectarse al servidor y descargar un archivo del mismo, para ello se ejecutan: **start_server** y **download-file**.



Como se puede ver en la imagen para descargar un archivo del lado del cliente se debe correr el archivo de Python **download-file.py**. Después, se aclara el protocolo a utilizar (TCP, SAW o GBN) y luego, en este caso, se introdujo un flag para indicar el archivo que se quiere descargar (para conocer los flags posibles correr en la consola la línea: "python3 download-file.py -h").

4.2. Subida de un archivo

En este caso se prueba que el cliente conectado al servidor pueda cargar un archivo al mismo, haciendo que este sea accesible para otros clientes.



Para conocer las distintas maneras en las que se puede correr el archivo **upload-file.py**, ejecutar en la consola de comandos "python3 upload-file.py -h".

4.3. Performance

Con el objetivo de comparar las distintas versiones de UDP implementadas y analizar como funcionan cuando hay pérdidas de paquetes se utilizaron disintos archivos de prueba: 100 kb, 2Mb y 5Mb. Para evaluar su funcionamiento con pérdidas de paquetes, se compararán los casos con 5 % de pérdidas, 10 % de pérdidas y 15 % de pérdidas utilizando la herramienta comcast. También, se analiza el caso de uso de TCP para poder analizar la efectividad de nuestras implementaciones.

4.3.1. Stop & wait

	100 kb	2Mb	5Mb
5 % pérdidas	3.6sec	2min	3.5min
10 % pérdidas	4sec	2.8min	4.9min
15 % pérdidas	9sec	3min	6.2min

4.3.2. Go Back N

	100 kb	2Mb	5Mb
5 % pérdidas	0.7sec	15sec	27sec
10 % pérdidas	1sec	15sec	36sec
15 % pérdidas	1.6sec	17sec	45sec

4.3.3. TCP

	100 kb	2Mb	5Mb
5 % pérdidas	0.004sec	0.07sec	0.2sec
10 % pérdidas	0.004sec	0.08sec	0.2sec
15 % pérdidas	0.004sec	0.09sec	0.22sec

4.3.4. Análisis

Teniendo en cuenta las mediciones obtenidas se puede observar que el protocolo más rápido es el protocolo TCP, esto se debe a que TCP ya implementa RDT. Dentro del uso del protocolo UDP, se observa que la versión de Go Back N es más rápida que Stop & wait, esto se debe a que esta aunque ambos esperan la confirmación del host receptor para mandar los próximos mensajes, Go Back N cuenta con una ventana (cantidad de paquetes a enviar) mayor que hace que la transferencia del archivo sea más veloz.

5. Preguntas a responder

5.1. Describa la arquitectura Cliente-Servidor

La arquitectura Cliente-Servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. El host servidor debe estar siempre activo (idealmente) para atender las consultas de los otros hosts (clientes). Los distintos clientes no pueden comunicarse directamente entre sí. Un cliente, inicia la conexión con el servidor, para ello el servidor debe contar con una dirección fija y conocida, denominada IP.

Algunos ejemplo de este tipo de arquitectura son: correo electrónico, juegos de red, navegadores web, etc.

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

La capa de aplicación define las aplicaciones de red y los servicios que puede utilizar un usuario, con ello determina como se comunican distintos procesos en diferentes sistemas finales. Esta capa ofrece a las aplicaciones la posibilidad de acceder a los servicios de las demás capas y define los protocolos que utilizan las aplicaciones para intercambiar datos. La misma define, los tipo de datos a intercambiar entre procesos, la sintaxis de los mensajes, sus campos y las reglas de cuando y cómo enviar o recibir mensajes.

5.3. Detalle el protocolo de aplicación desarrollado en este trabajo

El protocolo de aplicación desarrollado en este trabajo para implementar las funcionalidades de descarga y subida entre cliente y servidor conlleva un primer mensaje, donde el cliente, al conectarse, le envía un mensaje al servidor con la respectiva metadata del archivo a transferir. Luego, el servidor envía una confirmación sobre si la operación puede ser llevada a cabo y finalmente, se envía el archivo en sí.

La metadata envíada del archivo en el mensaje inicial consiste en:

- Tipo de operación: 1 byte indicando si la operación es de descarga o de subida (0 para descarga, otro para subida).
- Largo del path: 1 byte indicando el largo del path en bytes.
- Tamaño del archivo: 4 bytes indicando el largo del archivo en bytes (si el tipo de operación era subida).

La respuesta del servidor a este mensaje inicial depende del tipo de operación:

- En caso de una descarga el servidor envía al cliente un byte indicando el estado de la operación (éxito o error) y 4 bytes indicando el tamaño del archivo en bytes a envíar el cliente en caso de que el estado de la operación sea exitosa.
- En caso de una subida, el servidor le envía al cliente 1 byte indicando el estado de la operación (éxito (0) o error).

Finalmente, cualquiera sea quien envíe el archivo al otro host, el mismo se envía de a partes (chunks) de tamaño fijo. En cada chunk o paquete envíado se debe incorporar la siguiente metadata:

- 4 bytes indicando el offset del chunk en bytes.
- 4 bytes indicando el tamaño del chunk en bytes.
- "PAYLOAD_SIZE_BYTES"bytes de datos del archivo.

Para que host que envía sepa que el receptor recibió el paquete, este último envía un mensaje de acknowledge: 4 bytes indicando la cantidad total de bytes recibidos del archivo.

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? ¿Cuando es apropiado utilizar cada uno?

La capa de transporte permite la comunicación entre aplicaciones, lo hace multiplexando y demultiplexando mensajes. Dentro de esta capa, los protocolos UDP y TCP deben cumplir con esa funcionalidad.

UDP provee el servicio de entrega de paquetes a la capa de aplicación y ofrece un chequeo de errores utilizando el campo checksum en el header de sus paquetes. TCP, además de los servicios mencionados de UDP, ofrece Reliable Data Transfer, control de flujo y control de congestión.

UDP se caracteriza por poder perder paquetes y la posibilidade de que los mismos lleguen duplicados, además de que no se establece una conexión. TCP, en cambio, garantiza que no se pierdan paquetes, que no lleguen duplicados, que lleguen en orden y es de tipo *full duplex*, es decir que se establece una conexión entre cliente y servidor mediante la cual se pueden enviar y recibir mensajes en ambas direcciones, sin la conexión no se pueden enviar segmentos.

Es apropiado utilizar TCP cuando la transmisión de datos requiere de alta fiabilidad, dejando un poco de lado la velocidad, por ejemplo para el correo electrónico, la transferencia de archivos, etc. En cambio, es apropiado utilizar UDP cuando se requiere una comunicación rápida pero no tan confiable, esto puede ser útil por ejemplo para el streaming de multimedia, juegos online, telefonía, donde la velocidad de transmisión es esencial y donde la pérdida de paquetes no afecta en gran medida la experiencia del usuario.

6. Dificultades encontradas

- Sincronizar los mensajes envíados entre cliente y servidor y parsearlos correctamente.
- Conseguir que los mensajes envíados y recibidos no se desfasen al introducir pérdidas de paquetes.
- Cerrar correctamente todos los hilos abiertos por el servidor.
- Lograr que la transferencia del archivo no demore demasiado.

7. Conclusión

Tras haber implementado y analizado los distintos prótocolos TCP y UDP (Stop & wait y Go Back N) podemos concluir que la transferencia de archivos con una transferencia de datos confiable se ve totalmente facilitada por el protocolo TCP ya que el mismo ofrece confiabilidad en la recepción de los paquetes así como también garantiza el orden en que los mismos llegan. Esto facilita el diseño ya que desde la capa de aplicación no hay que tomar consideraciones extras para lograrlo.

Habiendo comparado los distintos protocolos, podemos notar que la implementación de UDP con Go Back N es más rápida que la versión con Stop & wait, esto se debe a que la primera implementación es menos susceptible a la pérdida de ACKs. También, se observó que la performance de nuestra implementación de TCP fue mejor que la de Go Back N, esto nos indica que aún hay cosas que se pueden mejorar para que sea comparable con TCP.

Finalmente, podemos concluir que el presente trabajo práctico nos ayudó a comprender mejor los temas de la materia además de brindarnos la oportunidad de familiarizarnos con distintas herramientas útiles o librerias para la implementación de los distintos protocolos así como también para el analisis de la performance en la transferencia de paquetes.