Introducción

El presente trabajo práctico tiene como objetivo la creación de una aplicación de red de arquitectura cliente-servidor que implementa la funcionalidad de transferencia de archivos mediante dos transferencias:

- UPLOAD (subida de un archivo del cliente hacia el servidor)
- DOWNLOAD (descarga de un archivo del servidor hacia el cliente).

Para ello, diseñamos e implementamos un protocolo de aplicación básico que especifica los mensajes intercambiados entre los distintos procesos.

Se utilizó la interfaz de sockets, el protocolo UDP de la capa de transporte, y protocolos de transferencia confiables implementados en la capa de aplicación. Los protocolos implementados son los denominados “Stop & Wait” [4] y “Selective Repeat” [5].

Por último, el servidor es capaz de procesar de manera concurrente la transferencia de archivos con múltiples clientes.

Hipótesis y suposiciones realizadas

Utilizaremos UDP como protocolo de la capa de transporte.

El cliente y el servidor se inician en dos terminales diferentes, ya sean dos hosts o dos instancias de una terminal, conectadas bajo una topología virtual a través de la herramienta Mininet [1].

Se definen los siguientes comportamientos por defecto para la interfaz por consola de la aplicación:

Implementación

La implementación se divide en 3 fases:

1. **Metadata Handshake**, donde se establece la conexión y se intercambia información sobre la misma.
2. **Correr Stop and Wait o Selective Repeat**, dependiendo de lo que se elija al ejecutar el comando.
3. **End connection**, donde el servidor y el cliente se ponen de acuerdo para terminar la conexión sin perder información.

Metadata Handshake: Nuestro protocolo maneja 2 paquetes para entablar una conexión: **Metadata** y **MetadataAck**. En ambos existen los campos *ActionType* (si la conexión es Upload o Download), *TotalFileSize* (donde se especifica el tamaño total del archivo, en cantidad bytes), y *Resource* (donde se especifica la ruta al recurso en el servidor que se está intentando descargar/subir). El paso a paso de ésta parte del protocolo se detalla en el gráfico 1: Remarcado en **Verde** se indica cómo se envían los paquetes si el cliente está en modo Upload, y el **Azul** se indica cómo se

envían los paquetes si el cliente está en modo Download. Es decir, cuando el cliente quiere descargar un archivo, envía un MetadataAck extra para así indicarle al servidor que está listo para recibir paquetes de datos.
En caso de que alguno de los paquetes se pierda, ambos cliente y servidor manejan un **timeout** y reenviarán el último paquete hasta cumplida la cantidad de intentos máxima (en cuyo caso, se dará por terminada la conexión por motivo de timeout).

*En **verde**: Metadata Handshake en caso de **Upload**.*
*En **azul**: Metadata Handshake en caso de **Download**.*

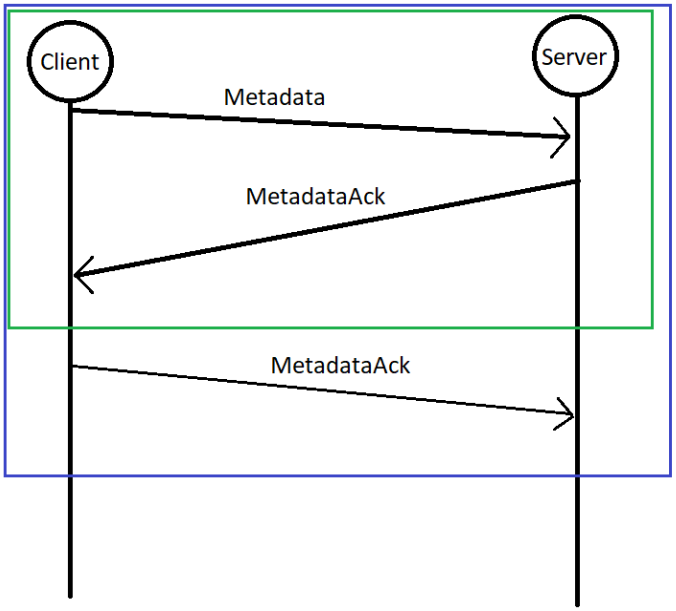

Gráfico 1: Metadata Handshake

Stop and Wait: Ésta parte del protocolo consiste en un intercambio de paquetes de datos, Data y DataAck, que se detallan a continuación:
Data:

- Campos:
  - Offset: Que representa a partir de qué posición en el stream de bytes comenzar copiar la data recibida
  - Block size: El tamaño del paquete enviado
  - Data: El paquete que se quiere enviar/recibir.

DataAck:

- Campos:
  - Offset: El offset del último paquete recibido

En este protocolo, una vez terminado el proceso del handshake, el nodo emisor inicia la transmisión de datos hacia el receptor con un paquete del tipo DATA. Posteriormente, el proceso se detiene en el lado del emisor mientras espera recibir una respuesta del tipo DATA_ACK y en caso de hacerlo, se repetirá este proceso hasta que se finalice el pasaje de información. En caso de que el nodo receptor

reciva múltiples instancias idénticas del mismo paquete, las desestima y responde con una DATA_ACK referente al último paquete recibido correctamente.

Pruebas

Las pruebas realizadas fueron:

Modo Stop and Wait

- Subida al servidor un archivo de texto de 5 bytes.
- Subida al servidor el mismo archivo de texto de 5 bytes (el servidor rechaza la conexión, al haber utilizado el mismo path del recurso).
- Descarga del archivo de texto de 5 bytes.
- Descarga del archivo de texto al mismo path local (del cliente), el cliente no inicia la conexión pues el archivo ya existe en su filesystem local.
- Subida al servidor de un archivo de 1 MB.
-

Preguntas a responder

1. Describa la arquitectura Cliente-Servidor.

Las redes de arquitectura Cliente-Servidor es un tipo de red que divide el sistema en dos partes. El servidor, que provee recursos o servicios y el cliente, que los consume. Este enfoque permite distribuir la carga de trabajo y promueve la colaboración en entornos distribuidos.

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

La función de un protocolo de capa de aplicación es facilitar la comunicación entre aplicaciones o sistemas distribuidos de una red. Establece reglas para facilitar el intercambio de información entre los distintos dispositivos conectados a la red.

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

Lorem ipsum.

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?

UDP

Servicios

Servicio no confiable: a diferencia de TCP, no introduce ningún retraso para establecer una conexión antes de enviar datos [2]. Esto significa que es más ligero y rápido, pero también menos confiable ya que los paquetes pueden perderse o llegar en orden incorrecto.
Servicio de datagramas independientes: cada paquete enviado a través de este protocolo se considera un datagrama independiente. Esto significa que cada paquete se trata como una unidad individual y no se garantiza que lleguen en orden o incluso que lleguen en absoluto.

Características

Sin control de flujo ni control de congestión: no proporciona mecanismos para controlar el flujo de datos ni para manejar la congestión en la red. Esto puede llevar a la pérdida de paquetes en redes congestionadas.
Encabezado simple: su encabezado es bastante simple, lo que lo hace eficiente en términos de procesamiento. Contiene solo cuatro campos: puerto de origen, puerto de destino, longitud y suma de verificación (checksum).
Transmisión unicast, multicast y broadcast: puede transmitir datos a una única dirección (unicast), a múltiples direcciones específicas (multicast) o a todas las direcciones en una red (broadcast), lo que lo hace flexible para una variedad de aplicaciones de red.

¿Cuándo es apropiado utilizarlo?

Es ideal para aplicaciones en las que la velocidad y la simplicidad son más importantes que la confiabilidad o la integridad de los datos. Ejemplos de tales aplicaciones incluyen video en tiempo real, voz sobre IP (VoIP), juegos en línea y transmisiones de medios, entre otros.

TCP

Características

TCP ofrece un servicio confiable, es decir, garantiza la entrega sin errores de datos mediante mecanismos como los mensajes de confirmación (ACK), la retransmisión de datos perdidos y el control de flujo

Características

TCP incluye características de seguridad como el establecimiento de una conexión mediante un proceso de triple confirmación (triple handshake), control de congestión para evitar la saturación de la red y la segmentación de paquetes, en caso de ser necesario.

¿Cuándo es apropiado utilizarlo?

TCP es adecuado para aplicaciones que requieren una comunicación confiable de datos, como la transferencia de archivos, la navegación web, los servicios de correo electrónico, entre otros.

Dificultades encontradas

Mininet: debido a que decidimos utilizar ésta herramienta para comprobar que nuestra aplicación efectivamente utiliza protocolos RDT, nos encontramos con el desafío de configurar una topología que cuente con pérdidas de paquetes. Ante la escasa documentación en internet (foros y bibliografía oficial), lograr que la topología implementada pierda paquetes requirió una iteración constante de prueba y error.

Inicio de la conexión (símil 3-way handshake): Dado que la conexión es no confiable en principio, necesitamos en una primera instancia compartir datos intrínsecos (en adelante llamados “metadatos”) al pedido del cliente con el servidor, para así poder continuar con alguno de los dos protocolos implementados. Estos metadatos incluyen:
El tipo de request del cliente (subir o descargar), el tamaño total del archivo a subir/descargar, el path al recurso (es decir, en el servidor, en qué path se va a guardar/leer lo que se está subiendo/descargando, respectivamente), y el primer bloque de datos, para no desaprovechar el envío de un paquete.

Manejo de errores y reenvío de mensajes:

Conclusión

En conclusión, la implementación exitosa de este protocolo nos ha demostrado la posibilidad de desarrollar protocolos confiables en la capa de aplicación, incluso utilizando como base de la capa de transporte un protocolo no confiable como lo es UDP. Esto nos ha permitido profundizar nuestra comprensión de los principios de la transmisión de información garantizando la entrega y recepción de datos, manejo de pérdida de paquetes, y particularmente el funcionamiento de los protocolos de Stop & Wait y Selective Repeat.

Referencias

[1] https://mininet.org/

[2] J.F. Kurose and K.W. Ross, *Computer networking: a top-down approach*, 7th Edition, (Pearson, 2017), p. 239 (chapter 3.3).

[4] J.F. Kurose and K.W. Ross, *Computer networking: a top-down approach*, 7th Edition, (Pearson, 2017), pp. 252-255 (chapter 3.4).
[5] J.F. Kurose and K.W. Ross, *Computer networking: a top-down approach*, 7th Edition, (Pearson, 2017), pp. 265-271 (chapter 3.4).