7543 - Introducción a los sistemas distribuidos

Trabajo Práctico n. 2:
Capa de transporte

Alumno	Padrón
Anca, Jorge	82399
Manzano, Matías	83425
Piazzi, Luciana	90638
Rozanec, Matías	97404

Índice

Objetivo

Introducción

El objetivo del presente trabajo es el estudio y la utilización de los protocolos de capa de transporte UDP y TCP para implementar un sistema de almacenamiento de archivos en la nube.

Para ello debimos implementar dos versiones del sistema utilizando los dos protocolos vistos en clase. Ambas versiones deberán asegurar la integridad de los archivos transferidos.

Cuadro comparativo TCP vs UDP vs QUIC

La responsabilidad fundamental de TCP , UDP y QUiC es extender el servicio de entrega de IP, que corren en la capa de red, entre dos end systems a un servicio entre dos procesos corriendo en dichos sistemas. Este servicio es denominado multiplexación - demultiplexación. Tanto UDP como TCP están implementados en el Sistema Operativo.

El protocolo UDP (User Datagram Protocol) corre en la capa de transporte y provee los siguientes servicios a las aplicaciones: envío de proceso a proceso y verificación de errores. Es el más rápido de los 3 ya que brinda menos servicios y le agrega un overhead menor a los datos a transferir.

El protocolo TCP (Transmission Control Protocol) también corre en la capa de transporte y provee los siguientes servicios a la aplicaciones:envío de proceso a proceso, verificación de errores, entrega confiable, y control de congestión.

Por último QUIC (Quick UDP Connections) es un protocolo desarrollado por Google en 2016 implementado en la capa de transporte sobre UDP y brinda básicamente los mismos servicios de TCP con alguna mejoras. Las dos ventajas principales de este protocolo sobre TCP son que es reduce el overhead de los mensajes y, al no estar implementado en el sistema operativo, es más rápido y sencillo hacer mejoras.

A continuación se agrega una tabla comparativa de los 3 protocolos:

	UDP	TCP	QUIC
Dónde corre	Capa de transporte	Capa de transporte	Capa de aplicación (corre sobre UDP)
Dónde se implementa	Kernel de los SO.	Kernel de los SO.	Espacio de aplicaciones
Confiabilidad	Sin garantía de entrega	Garantía de entrega	Con garantía de entrega
Conexión	Sin	Con handshake / con	Con handshake y conexión (implementado

	handshake/sin conexión	conexión	en la capa de aplicación)
Control de flujo	Sin control de flujo	Con control de flujo	Con control de flujo
Control de congestión	Sin control de congestión	Con control de congestión / con ventanas deslizantes	Con control de congestión
Control de integridad	Checksum	Checksum	Checksum
Seguridad	Ninguna	Se suele utilizar en conjunto con SSL + HTTPS	Los paquetes son encriptados individualmente y desencriptados en la capa QUIC
Latencia	Baja (no tiene handsake)	Alta (se agrega un RTT de handshake y el intercambio de claves, si se quiere securizar)	Media (en el mismo handshake se securiza la conexión)
Header size	8 bytes	20 bytes	8 bytes
Ejemplos de uso	DNS, Streaming	Http, Https, smtp, pop3, imap	Desde los browsers de chrome hacia los servers de google
Motivación	Envío de mensajes sin overhead.	Establecer mecanismos de garantías de entrega y control de flujo en un canal no confiable como es una red de computadoras	Dado la amplia implementación de los protocolos TCP y UDP y la imposibilidad de realizar un cambio en todos los dispositivos que las implementan y utilizan (ejemplo, sistemas operativos). Se consideró que fue mejor atacar el problema de las ineficiencias de TCP desde la capa de aplicación
Ventajas	Rápido.	Garantía de entrega con mecanismos de RTx	Todas la ventajas de TCP
	El socket se identifica con dos parámetros (scr_ip, scr_port)	Control de flujo determinado por la window que toma en cuenta el espacio en el buffer del cliente	No requiere cambiar el software de ningún SO.
	Requiere menor asignación de recursos, porque es conectionless	Control de congestión que permite no saturar la red.	Realiza la securización de la conexión en el handshake
			Posee un identificador de conexión, lo que permite mantener una conexión activa aún

			cuando el cliente cambie de IP (por ejemplo, al desconectar el wifi para conectarse al 3G)

Desventajas	No es confiable.	Lento (requiere más RTTs debido al handshake)	Probablemente sería más rápido si se implementara directamente en la capa de aplicación.
	No permite el control de flujo.	Overhead de 20 bytes en cada segmento enviado.	Dado que viaja menos información en el header de capa de transporte que en TCP, es más difícil el troubleshooting por parte de los manufacturadores de routers y proveedores de servicios
	Puede congestionar la red.		
	No mantiene conexión.		

Suposiciones y/o asunciones tomadas para realizar el trabajo

Durante el desarrollo del trabajo práctico se hicieron las siguientes suposiciones:

- no se tuvo en cuenta el desempeño de la aplicación
- tanto el cliente como el servidor tienen siempre memoria suficiente para almacenar los mensajes y los archivos
- no hay conexiones en paralelo

Explicación de la implementación del sistema utilizando TCP

Se mantuvo el esqueleto dado por la cátedra, en el que la implementación se ve dividida en dos: una aplicación cliente y un servidor. Como TCP tiene garantía, la implementación resultó más sencilla que la del sistema usando UDP.

Servidor TCP

El servidor crea un socket con la información de dirección y puertos recibidos por parámetro y espera a que los clientes se conecten. Una vez establecida la conexión espera un mensaje comunicando la operación que se desea realizar: U (upload) si se quiere subir un archivo o D (download) si se quiere descargar un archivo. Según sea el caso comienza a enviar o recibir el archivo. En ambos casos la principal verificación que se realiza es el de falla de conexión, en cuyo caso se lanza una excepción. Todos los casos posibles han sido

cuidadosamente considerados, de forma que el programa nunca termine de forma violenta, sino mediante manejo prolijo de la situación.

Cliente TCP

El cliente crea un socket y luego se intenta conectar al servidor según el puerto y la dirección Ip del mismo. Una vez establecida la conexión envía un mensaje comunicando la operación que se desea realizar, siguiendo el protocolo expuesto en la sección precedente. Al igual que el servidor, según sea el caso comienza a enviar o recibir el archivo. En ambos casos la principal verificación que se realiza es el de falla de conexión, en cuyo caso se lanza una excepción.

Explicación de la implementación del sistema utilizando UDP

La dificultad principal fue implementar el servicio de entrega confiable, es decir asegurar que todos los datos lleguen bien y en orden.

La mecánica general de las interacciones cliente - servidor es igual al caso de TCP. La diferencia radica en que en este caso se utilizaron objetos senders y receivers para el envío y recepción de información, que internamente garantizan que los mensajes son transmitidos incorruptiblemente y en orden.

Protocolo

Todos los mensajes están conformados siguiendo el siguiente protocolo:

- número de secuencia de longitud fija en 4 byes. Como esto es enviado como un string, los 4 bytes permiten enviar en cada posición un número entre 0 y 9, permitiendo así un número de secuencia máximo igual a 9999.
- checksum de 32 bytes. Si bien se podría haber aprovechado la información del header, a la hora de implementar esta funcionalidad fue más directo hacerlo de esta forma. Hizo también que la implementación y el control sean más explícitos.
- en el resto de los bits se envía el mensaje que será de longitud máxima CHUNK_SIZE, que es una constante definida en utils/constants.py

  Algo a considerar es que todos los mensajes se envían en binario. Es decir, no se pueden enviar strings. De querer hacerlo, habrá que hacer el encode() correspondiente, y el receiver deberá hacer el decode() para interpretar bien el mensaje. Dicho escenario se presenta en el trabajo únicamente en la comunicación de comandos.

  Además del mensaje de interés se envía un mensaje con número de secuencia 0 indicando la cantidad de segmentos que conforman el mensaje. Esto permite al receiver saber hasta cuándo debe estar recibiendo los segmentos.

Sender

Para el envío de mensajes se siguen los siguientes pasos:

- se calcula la cantidad de chunks en que habrá que partir el mensaje a enviar
- se crean los chunks necesarios
- se crea un vector donde se hace seguimiento de los segmentos confirmados.
  El vector mencionado tiene todos los números de secuencia de los mensajes a enviar; una vez recibido un ACK para un número de secuencia, se borra el elemento. De esta forma, se envían los segmentos en loop hasta que dicho vector esté vacío. Como se espera confirmación de todos los segmentos enviados, reenviando todas las veces que sea necesario, se puede estar seguro que serán transmitidos todos los segmentos.

Receiver

El receiver leerá los mensajes entrantes, guardando cada chunk recibido en un diccionario cuya clave indica el número de secuencia y el valor será el mensaje correspondiente a dicho número de secuencia. Una vez recibidos todos los segmentos, se juntan todos los valores del diccionario en orden, y se devuelve el mensaje formado.

Servidor UDP

A diferencia del caso anterior, luego de que el servidor crea un socket a partir de los parámetros recibidos,espera recibir mensajes sin tener que aceptar las conexiones. Una vez recibido el comando correspondiente a la operación a realizar se comienza a recibir o enviar los segmentos de datos.

Cliente UDP

A diferencia del caso de TCP, el cliente crea un socket, hace un bind con la dirección IP y puerto del servidor y envía mensajes, sin tener que establecer la conexión. Una vez enviado el comando correspondiente a la operación a realizar se comienza a recibir o enviar los segmentos de datos según corresponda.

Dificultades encontradas

Las mayores dificultades se han encontrado durante el setup de comcast debido a que la documentación es un tanto escueta y confusa.

En primer lugar, el default no verbose de la instalación rompe con la máxima de no dejar nunca al usuario sin que entienda qué está sucediendo. Correr el comando y que no haya ningún output fue muy desorientador.

En segundo lugar, para correr comcast hay que conocer el path de instalación, que se encuentra en algún lado de home/. Esto dicho de esta manera no parece nada raro, si no fuera porque en ningún tutorial se ejecuta llamando al path en home.

Una vez que se dedujo cómo ejecutar el comando, el grupo se encontró con exit status 1 (error). Esto se debió a que la “red” loopback llevaba el nombre lo, y no lo0.

Por último, para realizar el stop de comcast, hubo que agregar el device correspondiente, de lo contrario no se lograba el efecto deseado.

Una vez superados estos problemas, no hubo mayores inconvenientes con comcast. Los comandos a correr quedaron conformados entonces de la siguiente manera:

```
~/go/bin/comcast -device=lo --packet-loss=10%

~/go/bin/comcast --stop -device=lo
```

Conclusiones

Al desarrollar el trabajo práctico pudimos comprobar primero la mayor velocidad de UDP respecto de TCP. Esto se debe a que en UDP no hay establecimiento de la conexión (Three Way Handshake), no hay control de entrega y no hay control de congestión que limite el Throughput.

Por otra parte, verificamos que es posible implementar el servicio de garantía de entrega en la capa de aplicación utilizando UDP como protocolo de transporte. Asimismo pudimos comprobar que que dicho servicio que damos por hecho cuando utilizamos TCP no es trivial implementarlo y requiere un entendimiento de que realiza cada protocolo.

Implementar dicho servicio en UDP hizo que aumentaran los tiempos de respuesta por lo que una óptima implementación es muy importante.

Finalmente podríamos decir que implementar un servicio de garantía entrega sobre UDP tiene como ventaja sobre TCP la carencia de Three Way Handshake y el control de congestión. Por lo cual se reducen los tiempos de respuesta y se aumenta el Throughput.
Sin embargo es importante destacar que la mayor parte de los firewalls bloquean las conexiones UDP por lo cual no es una alternativa viable para redes públicas.