Redes de Computadoras

Facultad de Ingeniería - UdelaR

Curso 2017 - Grupo 48

Informe - Obligatorio 2

Autores:
Santiago Manduca C.I. 4.690.634-5
Felipe Hernández C.I. 4.327.276-3
Matías Rodríguez C.I. 4.908.877-8

11 de octubre de 2017

Índice

1. Objetivo **2**

2. Problema a resolver **2**

2.1. Descripción del servidor 2

2.2. Descripción del cliente 2

3. Solución propuesta **3**

3.1. Pregunta a) 3

3.2. Pregunta b) 7

4. Problemas Observados **9**

1. Objetivo

Aplicar los conceptos teóricos de capas de aplicación y transporte y la utilización de la API de sockets TCP y UDP

2. Problema a resolver

Se desea implementar un servicio de transmisión de video en línea (streaming) para usuarios en Internet. El mismo utilizará los protocolos TCP y UDP y buscará distribuir el vídeo entre todos los clientes conectados simultáneamente.
Además, se desea contar con una aplicación cliente para reproducir los vídeos recibidos.

2.1. Descripción del servidor

El servidor es la aplicación que permite hacer streaming de video. Este servicio tiene la capacidad de realizar la transmisión utilizando el protocolo TCP o UDP. Para el caso de TCP, el servidor acepta conexiones TCP en una dirección IP y puerto conocidos. Por cada conexión se realiza el streaming leyendo los datos a enviar desde una cámara web (o un archivo de vídeo). El streaming hacia un cliente finaliza cuando el cliente cierra la conexión.
Para el protocolo UDP el servidor espera por un mensaje de solicitud en una dirección IP y puerto conocidos y realiza el streaming hacia el cliente que hizo el pedido leyendo los datos a enviar desde una cámara web (o un archivo de vídeo). El cliente debe renovar la solicitud del streaming cada 30 segundos y el servidor debe finalizar la transmisión del video a un cliente si no ha recibido solicitudes por más de 90 segundos.

2.2. Descripción del cliente

El cliente realizará la reproducción de los streams recibidos. Para esto permitirá al usuario seleccionar el servicio TCP o UDP. Luego de seleccionada la opción el cliente se conecta al servidor y queda leyendo el stream, reproduciendo el video en pantalla. Para el caso UDP el cliente debe renovar su suscripción con el servidor cada 30 segundos.

3. Solución propuesta

3.1. Pregunta a)

Implemente el servidor y el cliente en un lenguaje de alto nivel a su elección. Deberá definir el mensaje de suscripción para el caso UDP.
Para el caso UDP puede asumir que el tamaño de un frame de video es siempre menor que el tamaño máximo de un segmento UDP.
Recuerde que la API de sockets TCP no es basada en mensajes sino en streams. Deberá tener esto en cuenta para el envío de los frames de video.

Para la reproducción de vídeo se utilizará la biblioteca OpenCV [1]. Esta biblioteca esta disponible para C/C++, Java y Python. En el Anexo se describen las funciones mínimas necesarias de OpenCV que deberá utilizar (para el lenguaje C++).
Se recomienda también revisar y utilizar como base el código de ejemplo de uso de sockets dado en el curso

Solución propuesta:

En primer lugar para la aplicación del lado del servidor se implementaron 3 estructuras de datos, las cuales se detallan a continuación.
Una clase de nombre GenericSocket y dos clases que heredan de la misma, denominadas UDPSocket y TCPSocket.
La decisión de implementar esta estructura es para operar con los sockets independientemente del protocolo de los mismos, TCP o UDP.
También, se consideró acorde generar una lista que contiene a los clientes, los cuales se agregan independientemente de si es un cliente que utiliza el protocolo TCP o UDP.
Se utilizó la biblioteca OpenCV con la función de capturar video de la cámara web o un archivo según correspondiese, además de definir una calidad en la cual codificar los frames y un formato, en este caso el formato JPEG.
Se crean hilos manejados mediante la biblioteca Thread que provee Python, para atender a los procedimientos acceptTCP, acceptUDP, sendFrame concurrentemen-

te.

El procedimiento acceptTCP se encarga del manejo de los nuevos clientes que utilicen TCP, agregando un nuevo cliente a la lista mencionada anteriormente. Análogamente, acceptUDP es el responsable de agregar nuevos clientes UDP a la lista, así como también actualizar el tiempo del último mensaje recibido desde el cliente.
Por último, la operación sendFrame lee de la captura de OpenCV, la codifica y para cada uno de los clientes envía un frame. Además si es un video, se deja en un bucle a efectos prácticos.

Para el manejo de un timer para clientes UDP se procede de la siguiente forma: en la operación acceptUDP cada vez que un cliente envía un mensaje de renovación de la suscripción, se refresca el tiempo del cliente seteandolo en el tiempo actual del servidor. Posteriormente, en la operación sendFrame se controla que el cliente haya renovado su suscripción en los últimos 90 segundos; si esto no se cumple no se envía el frame al cliente y se lo remueve de la lista.

Un aspecto importante de esta implementación es el Framing TCP.
Dado que el protocolo TCP consta de flujo (stream) de bytes, es necesario indicar cuándo comienza y termina un frame para así el cliente lo muestra correctamente. Para ello se implementó la técnica de length prefix, la cual consiste en enviar en primer lugar el largo de un frame, para luego enviar el payload correspondiente al mismo.
Para implementar la mencionada técnica, se utilizó la biblioteca struct de Python que permite el manejo de datos binarios sobre la red. Utiliza las denominadas Format Strings, que facilitan el empaquetado y desempaquetado de datos.
Para enviar el largo del frame, se utiliza la operación struct.pack, a la cual se le pasan como parámetros una Format String y el largo del frame. En este caso la Format String enviada es '$> I$', que indica que el formato en que se almacen los bytes sea big-endian ($>$) y que sea un unsigned integer. Del lado del cliente, existe una llamada a struct.unpack con la misma flag y el largo 4, dado que es el largo de un unsigned integer.

En la aplicación del lado del cliente se implementaron procedimientos para recibir

frames mediante TCP y UDP, recv_one_message y recv_one_messageUDP respectivamente.
La primera de ellas desempaqueta el largo del frame, para luego recibir el payload llamando al procedimiento recvall pasándole dicho largo como parámetro. Esto es necesario debido al Framing TCP que se implementa en el lado del servidor, explicado anteriormente. Para el caso de un cliente UDP, su operación de recepción no hace más que recibir el payload directamente. También para dicho caso se envía un mensaje de renovación de suscripción cada 30 segundos al servidor.

Forma de llamada Cliente:
$./client.py\ ServerIP\ TypePort = UDP/TCP\ NumberPort$
Forma de llamada Server:
$./server.py\ file(0\ for\ camera)\ ip\_addr\ tcp\_port\ udp\_port$

Se realizaron dos capturas en Wireshark de los siguientes escenarios, a modo de comparación entre TCP y UDP:

- Dos clientes uno TCP y otro UDP que abandonan la suscripción de streaming del servidor.
- El servidor corta el streaming.

En el primer caso se observa que filtrando por UDP (udp && ip.addr == 192.186.1.119), el envío de paquetes es el esperado. Se destaca en primer lugar, el "HELLO"de suscripción que tiene payload de tamaño 5, lo cual sucede cada vez que renueva la suscripción, es decir, cada 30 segundos. También hubo segmentos UDP que no llegaron a su destino, marcados por un mensaje ICMP de "Destination unreachable (Port unreacheable)". Esto refleja la diferencia con el protocolo TCP en cuanto a posibles pérdidas de frames.

Para el cliente TCP, el flujo es el esperado. Al filtrar los segmentos en Wireshark (tcp && ip.addr == 192.168.1.114) se destacan, el 3-way-handshake inicial para establecer la conexión, durante la misma existe el flujo de bytes esperado, y luego al finalizar la conexión se envía desde el cliente un segmento con la flag FIN. En el segundo escenario, se destaca en ambos protocolos el mensaje de finalización

indicado por un "BYE", que señala a los clientes que deben finalizar su ejecución con el correcto manejo de sockets a nivel de implementación.
A su vez se observó en ambos escenarios para ambos protocolos la existencia de paquetes con un mensaje que contiene "JFIF", que indica que efectivamente se está enviando un frame JPEG.[5]
En la práctica se observa que el cliente UDP está más adelantado que el TCP, sin embargo, a simple vista esta diferencia es mínima. Además tampoco es notoria la perdida de segmentos en UDP.
En estas pruebas, la dirección IP 192.168.1.117 corresponde al servidor, 192.168.1.114 el cliente TCP y 192.168.1.119 el cliente UDP.

3.2. Pregunta b)

Explique detalladamente como debería cambiar su solución si el supuesto anterior (el tamaño de un frame de vídeo es siempre menor que el tamaño máximo de un segmento UDP) deja de ser válido. Su propuesta debe ser robusta a la pérdida o reordenamiento de paquetes. Justifique.

Solución propuesta:

En primer lugar, dado que el tamaño de un frame no siempre es menor que el tamaño de un segmento UDP, se considera enviar la cantidad de segmentos necesarios para contener el tamaño de un frame.
Bajo la suposición de que se está haciendo un streaming con resolución de video 4K, se pueden esperar frames de aproxidamente 16MB. Dado que un segmento UDP tiene como máximo un payload de 65007, entonces cada frame necesitaría aproxidamente 256 ($16MB/2^{16}$) segmentos UDP. Entonces, se necesita 1 byte para indicar el número de secuencia.

Este número se utiliza para gestionar el reordenamiento y la pérdida de segmentos, implementando una técnica similar al control de congestión en TCP. Para esto, se dispone de un buffer del lado del servidor que recibe los ACK de los segmentos y otro buffer del lado del cliente que almacena segmentos con sus respectivos números de secuencia.
Además del lado del servidor se implementa un timer para cada segmento enviado y comienza un nuevo frame una vez que recibe todos los ACK correspondientes a los números de secuencia enviados.

Del lado cliente, se muestra el frame una vez que en su buffer se encuentran la misma cantidad de segmentos que el servidor le envió en primera instancia. Como contrapartida a esta técnica observamos que si el primer segmento que envía el servidor se pierde, evidentemente el cliente no sabe la cantidad de segmentos que recibirá. Esto se puede solucionar con una flag que indique que dicho segmento es de hecho el que indica la cantidad de segmentos a enviar.
De esta forma, del lado del cliente no se almacenarían segmentos hasta que no se

reciba el segmento que contiene la flag mencionada.

4. Problemas Observados

Al cerrar el servidor con algún cliente TCP conectado, al intentar reiniciarlo el puerto se encontraba en uso.
Esto se debe a que a pesar de cerrar el socket, el sistema operativo del servidor posee un valor denominado MSL(Maximum Segment Lifetime) definido como el tiempo en el que un paquete TCP puede existir en el sistema.
Para esto se utilizó la bandera SO_REUSEADDR[4], que permite volver a utilizar la dirección.

Referencias

[1] Open Source Computer Vision Library. opencv.org

[2] Documentación de OpenCV. http://docs.opencv.org/2.4/index.html

[3] Imagen ISO de Lubuntu 17.04.
http://cdimage.ubuntu.com/lubuntu/releases/17.04/release/lubuntu-17.04-desktop-i386.iso

[4] http://man7.org/linux/man-pages/man7/socket.7.html

[5] https://tools.ietf.org/rfc/rfc2035.txt