Redes de Computadoras

Curso 2017

Grupo 76

Informe - Obligatorio 02

Autores:
Joaquín Villar
Federico Luongo
Federico Godoy

Supervisores:
Martín Giachino
Jorge Visca

October 8, 2017

Contents

1 Implementación Servidor y Cliente

1.1 Introducción

En este informe se presentarán los principales puntos tomados en cuenta por el equipo a la hora de implementar una aplicación de streaming de vídeo desde un servidor a múltiples clientes. La aplicación permite al usuario seleccionar el servicio TCP o UDP. Luego de seleccionada la opción el cliente se conecta al servidor y este es capaz de emitir el contenido de vídeo a cada uno de los clientes conectados como se puede observar en la siguiente Figura 1.

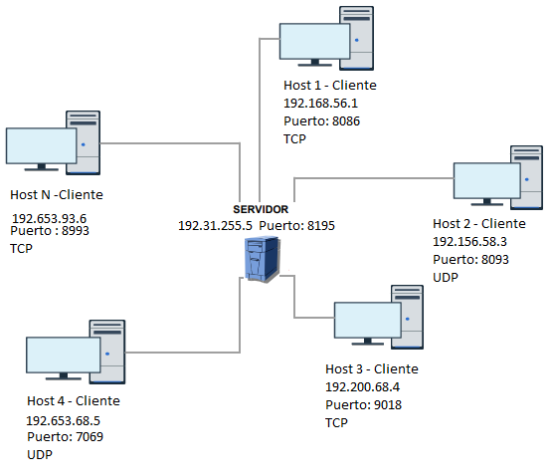

Figure 1: Imagen de ejemplo.

El servidor acepta conexiones TCP en una dirección IP y puerto conocidos y también recepciona mensajes de subscripción UDP para dichos clientes que luego de esto comienzan a recibir la imagen de video.

1.2 Servidor

Aplicación implementada en Python y se hace uso de la librería OpenCV. El servidor para ser ejecutado debe recibir los siguientes parámetros:
python Server.py (dirección IP), (puerto), (0 = cámara web o ubicación dentro de la carpeta del servidor del vídeo).
La implementación del servidor cuenta con un procedimiento principal y dos hi-

los. En el procedimiento principal el cual se encarga de recibir las imágenes a partir de la api de CV2 y luego a esta imagen se le baja la resolución hasta que tenga un tamaño menor al de un datagrama UDP. Luego se recorre un diccionario donde se guardan las conexiones TCP y se envía a estos cada frame, en este envio antes del frame convertido a string se concatena el largo del string a enviar en un tamaño fijo "L" (unsigned long integer) para poder ser identificado en el cliente de manera correcta. Luego se recorre la estructura de clientes UDP pero en vez de las conexiones se guarda ip y puerto correspondiente de cada cliente. Realizamos la implementación de esta forma para asegurarnos de que cada cliente reciba los mismos frames. Se puede observar que en esta implementación se esta dando una prioridad a TCP en el orden en que se envían los paquetes aunque desde el punto de vista computacional puede ser mínima, esta existe.
En los hilos se puede encontrar un hilo para TCP el cual se encarga de aceptar las conexiones nuevas TCP y las agrega a un diccionario en el cual se guarda la conexión y la dirección luego se hace un merge con el diccionario de las conexiones activas TCP. En el caso del hilo UDP también se encarga de recibir nuevos mensajes de solicitud UDP pero a diferencia de TCP en el diccionario se guarda la dirección/puerto y la fecha hora en la que se recibió esta solicitud, luego en el momento de hacer el merge de este diccionario (nuevas conexiones) con el diccionario de conexiones activas se controla si no han sido superados los 90 segundos sin recibir un keep-alive de dicho cliente. En ambos casos tanto TCP como UDP los diccionarios de nuevas conexiones son variables que se comparten entre el procedimiento principal y los hilos de tal forma de que se pueda utilizar en ambos lados.

1.3 Cliente

También se encuentra programado en Python y brinda la interfaz a los distintos clientes para porder comunicarse con el Servidor y recibir la imagen de video. En el lado del cliente también se requiere de una serie de parámetros estos son los siguientes:
python Client.py (dirección del servidor), (puerto del servidor), (UDP o TCP).
En este depende de si se quiere ejecutar utilizando UDP o TCP tiene un comportamiento diferente. En el caso de TCP se genera una conexión con el servidor y

dentro de un while se ejecuta un recive donde en el caso de TCP primero se realiza un while donde se ejecuta hasta recibir un tamaño "L" (unsigned long integer) donde dentro de este se tiene el largo del string del frame a recibir, luego se ejecuta otro while por el largo del que se había recibido antes hasta recibir todo el frame, este se agranda para que sea visible y se imprime en pantalla. En el caso de UDP al no dividirse los envíos en varios paquetes, se recibe un frame y este se agrande e imprime en pantalla, para poder mantener activo este cliente en el caso de UDP se genera un hilo que cada 30 segundos le envía al servidor un mensaje UDP donde se inicia o re confirma la suscripción.

1.4 Pruebas realizadas

En esta sección se documentan los casos de prueba más relevantes que se realizaron al sistema implementado y fueron de ayuda para encontrar ciertos defectos para ser corregidos.
Estas pruebas se ejecutaron en un habiente donde la congestión de la red es relativamente baja con una velocidad de red alta (fibra óptica), para homogeneizar las variables entre los procesos de envío TCP y UDP se decidió que en ambos se envíe la imagen con menor resolución y en ambos clientes esta imagen se agranda para que pueda ser visible. En diferencia entre ambos se puede observar que en el servidor primero se envía el frame a TCP y luego a UDP, además en el cliente en el caso de UDP se corre un hilo en paralelo y en TCP ademas de enviarse el frame se envía concatenado el tamaño de este lo que genera que el cliente tienen que recibir mas datos e iterar sobre estos. Esta fue la base con la que se ejecutaron la prueba a no ser de que la prueba especifique algún cambio para probar algo en especifico. Para comprobar algunos aspectos del funcionamiento se utilizó la herramienta WireSahrk y pudo observarse un mayor trafico de paquetes por parte de clientes TCP con respecto a UDP, dado que para el primero encontramos los mensajes ACK y eventuales retransmisiones de paquetes perdidos.

1.4.1 Caso 1

En este caso se prueba la conexión de dos clientes que seleccionaron la opción TCP y otro dos UDP. Una vez levantado el servidor, este queda a la espera de solicitudes de conexión TCP entrantes y mensajes de solicitudes UDP. Una vez ejecutados los

programas clientes en los cuales se indica la dirección IP y puerto del servidor junto con la opción TCP o UDP, a medida que se establecen las conexiones TCP y las solicitudes UDP son recibidas el servidor comienza a hacer el envió de la imagen de vídeo a los respectivos clientes. En este caso se mantuvo Servidor y Clientes funcionando durante un tiempo de 5 minutos aproximadamente sin obtener ningún error. Se puede observar cierto retraso y mínimos saltos en la imagen en el caso de clientes UDP que se incrementan con el pasar del tiempo de streaming.

1.4.2 Caso 2

En segundo lugar se realizó una prueba sobre los clientes UDP y el correcto funcionamiento de los mensajes "keep-alive" por parte de los clientes. Una vez levantado el servidor y reconocidas las solicitudes de los clientes, el vídeo comienza a ser recibido por parte de los clientes. Del lado del servidor se muestran las solicitudes iniciales de dos clientes distintos, con el transcurso del tiempo se muestran los mensajes de renovación de solicitud de cada cliente. En determinado momento se procede a cerrar uno de los clientes y se observa que pasados 90 segundos desde el último mensaje de renovación de dicho cliente, en el servidor se indica que se ha desconectado el cliente porque dejaron de recibirse los mensajes de renovación por parte de dicho cliente. Mientras tanto el otro cliente continúa recibiendo el stream de vídeo y a su vez renovando su suscripción mediante el envío de los mensajes cada 30 segundos.

1.4.3 Caso 3

En este caso se verificó la respuesta del servidor frente a un mayor número de clientes conectados. Se ejecutaron 4 clientes UDP y 4 clientes TCP tales que reciben imagen de vídeo desde el mismo servidor. En este caso todos los clientes reciben el vídeo de forma correcta, con cierto retardo en los casos de UDP como se había observado en el caso anterior. En esta prueba luego de cierto tiempo los mensajes de "keep-alive" UDP incrementan su demora y en algunos casos se pierden. Esto lleva a que para algunos clientes no se concrete ningún envió de renovación de suscripción durante 90 segundos y como resultado ese cliente deja de recibir imagen de vídeo y se cierra.

1.4.4 Caso 4

Se verificó que el estado de los clientes conectados ante la baja del servidor sea consistente. Una vez que el servidor y varios clientes se encuentren recibiendo el stream de vídeo, ya sea clientes UDP y TCP. En determinado momento se cerró el programa servidor y se puede ver que los clientes dejan de ejecutarse, en el caso de TCP el programa se cierra. Por otro lado UDP queda en un estado que en el caso que el servidor vuelva a estar activo, dichos clientes continúan recibiendo imagen de vídeo. Se entiende que los clientes TCP cierran sus conexiones y no vuelven a re-conectarse si el servidor vuelve a estar activo, esto es discutible ya que podría implementarse un procedimiento que permita intentar la re-conexión determinado tiempo después. Se comprobó que ante el cierre del servidor mediante el comando Ctrl + c se liberan los recursos asociado al socket TCP. Este modo de funcionamiento de decidió dado que al levantar el servidor nuevamente los clientes UDP vuelven a su funcionamiento normal y en el caso de los clientes tcp si se quisieran volver a conectar al servidor no tienen ningún problema.

1.4.5 Caso 5

Para este caso se probo enviar distinta resolución de imagen para TCP y UDP dado que para TCP no hay restricción de tamaño al enviar un frame, en esta prueba ante la realidad planteada originariamente se observaban dos nuevas variables en TCP se envían mas datos por frame pero en el cliente TCP no se agranda la imagen. En respuesta a este cambio se observo que TCP se comportaba un poco mas lento pero seguía siendo superior al rendimiento de UDP.

1.4.6 Caso 6

Como último caso se realizo la prueba de cerrar los clientes UDP y TCP durante la ejecución del stream. Para UDP se liberan los recursos asociados al socket y en el caso de TCP se inicia la desconexión por parte del cliente, el servidor interpreta dicha solicitud y termina la desconexión. En este caso se verificó que el servidor continúa siendo capaz de recibir nuevas solicitudes de otros clientes luego de realizar alguna desconexión.

1.4.7 Comparación TCP vs UDP

Dadas las pruebas y la forma en la cual esta programada la solución y las situaciones por las cual se ejecutan las pruebas se puede notar un mejor rendimiento de TCP ante UDP cuando se trata de una red poco congestionada por ejemplo un cliente TCP y un UDP. Cuando empezamos a agregar clientes tanto TCP como UDP se puede apreciar que la velocidad de transmisión como el retardo y saltos en el caso de UDP empiezan a homogeneizarse con el comportamiento de TCP donde este disminuye su velocidad de transmisión. Esto se debe a que este protocolo tiene un control de congestión y disminuye su utilización de la red lo que genera que se observe un comportamiento similar en ambos. Para poder observar la diferencia se hizo una prueba en la que el envío del video hacia TCP no se bajaba la resolución lo que llevaba a que el protocolo tenga que enviar mayor cantidad de datos pero no tenia que agrandar la imagen en el cliente, en esa prueba se podía observar que ademas de temer mejor rendimiento TCP que UDP la calidad de la imagen era mejor por lo que nos llevaba a la conclusión que en nuestro caso basándonos en una red local que en un principio esta poco congestionada y la forma en que se programo cada instancia el protocolo TCP obtuvo un mejor rendimiento que en UDP. Se destaca que ante casos de congestión algunos clientes UDP no llegan a trasmitir el sus mensajes de keep alive en tiempo y forma y la conexión se cierra mientras que TCP continua con funcionamiento normal.

2 Solución planteada para el caso UDP

En el caso de no poder asumir que el tamaño de un frame de vídeo sea menor que el tamaño máximo de un segmento UDP presenta el problema que necesariamente tiene que resolverse mediante la segmentación de dichos frames para que puedan ser enviados a través de los segmentos y luego de recibidos por los clientes puedan formar nuevamente el frame para ser mostrado correctamente. Como sabemos UDP proporciona un servicio de transferencia de datos no fiable; es decir, cuando un proceso envía un frame a un socket UDP, el protocolo UDP no ofrece ninguna garantía de que la información enviada hacia el proceso receptor llegue en el mismo orden de salida lo cual en este caso implicaría que la imagen mostrada a los clientes no sería óptima. Para esto se propone modificar nuestra solución implementada

de la siguiente manera. Primero que nada del lado del servidor se fragmenta cada frame de video obtenido desde la cámara o archivo de forma tal que ese fragmento no supere el tamaño fijo de los datagramas UDP, a dicho fragmento se le asigna un número de secuencia el cual es incluido en el datagrama al principio del campo de datos. Junto con el número de secuencia del fragmento se agrega un segundo número de secuencia que identifique a que frame pertenece. En el procesamiento de un frame del lado del servidor es necesario indicar cual es el ultimo fragmento de un frame para que este puede ser mostrado en la pantalla del cliente, para esto se indica al final del campo de datos un numero de secuencia especial fijo. Del lado del cliente se cuenta con una estructura capaz de guardar los fragmentos del último frame que esté siendo recibido y procesado, en el caso de que uno de estos fragmentos se pierda se descartan los restantes y en consecuencia la aplicación no muestra dicho frame. Luego de esto una vez identificado el ultimo fragmento del frame anterior se comienza a guardar en la estructura los fragmentos del siguiente frame para ser mostrado en pantalla. Cabe aclarar que para dar un fragmento por perdido se cuenta con un temporizador que establece el tiempo máximo de espera por un fragmento, dada nuestra realidad el temporizador debe estar seteado un valor pequeño dado que el stream de video no puede perder continuidad en su emisión. Se destaca que en este caso ante la recepción de un frame incompleto o fuera de orden los fragmentos son descartados por lo tanto se puede afirmar que la aplicación tolera la pérdida de frames y continúa funcionando con los frames que son recibidos completa y correctamente. Otra posible implementación podría ser que en vez de implementar un timer en el cliente cuando se complete un frame todos los frames anteriores a este sean descartados y este sea visualizado.

References

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

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

[3] Socket Programming HOWTO - https://docs.python.org/2/howto/sockets.html

[4] Redes de computadoras - Un enfoque descendente - Kurosse y Ross - 5ta Edición - https://docs.python.org/2/howto/sockets.html