Redes de Comunicaciones II

Práctica 2

RTSP, RTP, RTCP

Diego González Sánchez

Miguel Gordo García

24-IV-2012

Ejercicio Obligatorio

1.1 Introducción

En este ejercicio se pide la implementación de dos programas escritos en C que realicen las funciones de un servidor multimedia que distribuye video almacenado y de un cliente que desea conectarse a dicho servidor y solicita la reproducción de un video. El sistema debe implementar el protocolo RTP para la transmisión de los datos por UDP, el protocolo RTSP para la solicitud de comandos relativos a multimedia (conexión, preparación de la transmisión, iniciar transmisión, pausar temporalmente el envío de datos y desconexión) y por último el protocolo RTCP para recopilar estadísticas relativas a las pérdidas de paquetes y al jitter entre paquetes. El servidor dispone de un video cuyo audio y video están previamente divididos en dos archivos independientes. La transmisión del audio y video se hace por canales separados. Se espera que el servidor sólo acepte una conexión simultánea. Para la reproducción se invoca el programa gstreamer, que se encarga por su cuenta de sincronizar audio y video

1.2 Características del Proyecto

Nuestra implementación usa la interfaz en GTK proporcionada para el cliente. Dicha interfaz recibe sus argumentos en el cuadro de texto destinado a tal efecto en la ventana de la interfaz. Los argumentos a introducir son. Tras invocar el cliente desde terminal con ./video, los argumentos a introducir son

<XXX.XXX.XXX.XXX> <puerto> <directorio>

Donde <XXX.XXX.XX> es la dirección IP del servidor, <puerto> es el número de puerto donde el servidor espera conexiones y <directorio> es el nombre de la carpeta donde el servidor guarda el video, en nuestro caso, dir.

Para instanciar el servidor se debe introducir desde terminal:

./servidor <puerto>

Donde <puerto> es el puerto donde se desea esperar conexiones.

Como se especifica en su RFC el protocolo RTP está implementado sobre TCP para el control de la conexión, mientras que RTCP y RTP corren sobre sockets UDP. Estos protocolos son independientes entre sí y están implementados en sus respectivas librerías. Las funciones de sockets comunes usan nuestra librería socket_funcs que encubre las llamadas a las inicializaciones usuales de sockets TCP y UDP, respectivamente.

1.3 Desarrollo técnico del proyecto

Un aviso a tener en cuenta a la hora de probar la práctica es que los botones para la conexión han de pulsarse escrupulosamente en el orden establecido: Connect, Setup, Play. En otro caso la ventana saltará con error, pero NO se cerrará la conexión con el servidor. Es decir, hay posibilidad de acto seguido pulsar el botón correcto y continuar con la operación.

Se ruega encarecidamente consultar las estrucutras utilizadas en la documentación técnica anexa para comprender más a fondo la implementación de los protocolos utilizados, especialmente RTCP y RTSP.

La correcta liberación de los recursos asignados sólo se produce al enviar Teardown tras iniciar una transmisión correcta del video. En pos de una mayor simplicidad en el servidor se esperan los mensajes en el orden correcto para iniciar la transmisión de vídeo y audio y no se espera activamente en cualquier momento la llegada del teardown. Es importante recordar esto para no dejar recursos sin liberar.

Los protocolos subyacentes no están completamente implementados: hay campos de las estructuras y otros protocolos subyacentes (como el NTP) que no se han implementado ni se han tenido en cuenta en la práctica. El siguiente apartado proporciona más información en este aspecto.

La práctica consta por el lado del servidor con un total de 8 hilos, dos para escucha y envío de RTSP, dos hilos RTP para enviar el audio y el video y 4 hilos RTCP, dos para el audio y dos para el vídeo, de envío y escucha respectivamente. Cuando, por ejemplo, el cliente pide Pause, el hilo de control de RTSP pone unas variables globales para la transmisión al estado “Pausado”, provocando que el flujo de los hilos se altere y acabe en un semáforo a la espera de que el cliente reanude la reproducción con un nuevo Play. En este momento, el control de RTSP cambia de nuevo las variables globales y levanta los semáforos para que la transmisión de paquetes continúe por el punto donde se dejó.

En el cliente el número de hilos también es 8: el principal, dos de escucha de RTP, uno de control del Gstreamer, y 4 de escucha y envio de RTCP. Nuestra implementación de RTCP gestiona

el tiempo de espera entre paquetes en el servidor, de manera que si el cliente informa de estar perdiendo un elevado número de paquetes, el servidor aumentará el delay entre los mismos, mientras que si ve que todo va bien lo irá disminuyendo progresivamente.

En el cliente, la solicitud de un Pause no pausa la reproducción, sino el envío de paquetes por parte del servidor, lo que causará que tarde o temprano la reproducción del video pare por falta de paquetes. Cuando se reanude la reproducción, se reanuda inmediatamente puesto que a la velocidad de envío es probable que el pipe se llene y se genere un cuello de botella, causando perdida de paquetes por desbordamiento del socket. Tampoco se hace espera al inicio del Play, puesto que al fin y al cabo el pipe sólo tiene 64k de capacidad. Se podría haber pausado la reproducción poniendo el estado del pipe del Gstreamer.

Teardown deja al servidor presto para recibir nuevas solicitudes de video, y acaba con la ejecución del cliente. Nuestros ejemplos se han hecho con la versión de 480p de Big Buck Bunny. Otro aspecto importante del servidor es que en la ejecución se espera que el audio y el video estén ya partidos de antemano: el programa no realiza esta funcionalidad. En nuestro caso hemos partido Big Buck Bunny con el programa “split” suministrado en los ejemplos. También es importante que en el directorio donde se encuentren el audio y el video se halle un documento en .txt llamado config que contenga un pequeño texto en este formato:

audio 0 RTP/AVP 0
control: rtsp: /dir/audio.ogg
video 0 RTP/AVP 26
control: rtsp: /dir/video.ogg

De esta manera la respuesta al describe del cliente coge de aquí los datos necesarios para los campos del mensaje (rutas del archivo, etc..).

Para el cálculo del jitter se nos ha presentado el problema de no tener el timestamp del momento en que se enviaron los paquetes RTP (el campo en la cabecera es demasiado pequeño para un timestamp convencional). Como es muy difícil hacer una desviación típica con datos que no conocemos, hemos optado por calcular la media de los retrasos entre paquetes, y eso es lo que se escribe, entre otras cosas, en el fichero de estadísitcas del servidor.

1.4 Funciones

La documentación funcional se genera automáticamente ejecutando make doc y se elimina con make clean.

1.5 Compilación

Simplemente ejecutar el comando make en la consola

1.6 Resultados y Posibles Mejoras

En el directorio de la práctica se ha incluido una carpeta llamada gráficas que contiene los resultados obtenidos en la medición de paquetes perdidos y jitter en tres pruebas distintas:

- Audio a 2000 Bps, Video a 1000Bps, Pipe No-bloqueante.
- Audio a 1000 Bps, Video a 800Bps, Pipe No-bloqueante
- Audio a 1000 Bps, Video a 1000 Bps, Pipe Bloqueante

Nuestro programa adolece de un problema, que es el tamaño del pipe (64kb). Hemos investigado como cambiarlo pero la solución consistía en cambiar el número de buffers que el sistema operativo reserva para el pipe, tocar ficheros del sistema operativo...y desistimos. Es cierto que unos compañeros han usado una estrucutra de cola dinámica de Gtk creo para no tener nuestro problema, pero por cuestiones de tiempo y esfuerzo ya gastado no lo cambiamos al final.

Esa es la razón de las abruptas subidas y bajadas. Cuando el pipe se llena gstreamer tarda en consumir video y el pipe tarda en vaciarse, causando pérdida de paquetes por desbordamiento del socket. Cuando el pipe se bloquea se produce un pico de pérdidas y conforme pasa el tiempo el problema escala y se hace inmanejable. Pensamos que poner el pipe no bloqueante y directamente desechar el paquete del socket daría mejores resultados porque en teoría no debería desbordarse el socket, sin embargo la calidad del video es peor, con más trompicones.

El tamaño del paquete lo hemos dejado al final en 1000 Bytes y espera bloqueante, por ser el que mejor calidad aparente da al vídeo, aunque por los problemas mencionados, la reproducción acaba volviéndose inaceptable. Las gráficas las hemos dejado en un directorio dentro de la práctica, el nombre de cada una de ellas explica su contenido. La nomenclatura seguida es: <Video o audio> <tamaño del paquete de audio> <tamaño del paquete de video>. Una b al final indica bloqueante

De todas las pruebas realizadas poner el audio a 1000 Bytes y el video a 800 Bytes es la que

mejores resultados ha dado, aunque el momento en que el pipe se llena los paquetes se pierden y el problema escala hasta ser irresoluble. Incluso intentamos hacer un flush del pipe, para en estos casos intentar tirar todo y volver a empezar, pero no lo conseguimos.

La medición del jitter entre paquetes da buenos resultados salvo cuando empieza el problema del llenado del pipe/desbordamiento del socket, que es cuando bien se dispara por las nubes o comienza a oscilar con grandes variaciones.

La mejora clara que debe hacerse es la de cambiar la estrucutra del pipe por otra cola de más capacidad, dinámica, que pueda aceptar más video y audio. Con una capacidad tan limitada y sin saber la velocidad de consumo de media por parte de gstreamer parece la solución más rápida y desde luego más aceptable dado que de 4Gb de memoria, 64Kb parece un mal chiste por parte del sistema operativo.

Hay varios errores que se han dejado en la práctica desde la entrega parcial sin ser corregidos por falta de tiempo, y estos son algunos comentarios, y los detalles referentes al control de errores. Algunos se han cambiado, otros se han dejado como estaban por economía de nuestro tiempo, ya que esta práctica se puede perfeccionar ad infinitum, pero no es el propósito principal y nuestro trabajo es mucho y tiempo limitado. Somos conscientes de ellos y los asumimos.

Diego González Sánchez
Miguel Gordo García