Trabajo de Programación
Video Streaming con RTSP y RTP
IT3436 Redes Multimedia
Prof. Candy E. Sansores

El Código

En esta práctica de laboratorio, se implementará un servidor de streaming de vídeo y el cliente que se comunican utilizando el protocolo de transmisión en tiempo real (RTSP) y enviar datos utilizando el Protocolo de transferencia en tiempo real (RTP). La tarea es poner en práctica el protocolo RTSP en el cliente e implementar el empaquetado RTP en el servidor.
Se proporciona el código que implementa el protocolo RTSP en el servidor, el desempaquetado de RTP en el cliente, y se encarga de mostrar el video transmitido. No es necesario tocar este código.

Clases

Cliente
Esta clase implementa el cliente y la interfaz de usuario que se utiliza para enviar comandos RTSP y que se utiliza para mostrar el vídeo. A continuación se muestra un ejemplo de la interfaz se. *Se tendrán que implementar las acciones que se realizan cuando se pulsan los botones.*

Servidor
Esta clase implementa el servidor que responde a las peticiones RTSP y que realiza el streaming del vídeo. La interacción RTSP ya está implementada y el servidor llama a las rutinas de la clase RTPpacket para empaquetar los datos de vídeo. No es necesario modificar esta clase.

RTPpacket
Esta clase se utiliza para manejar los paquetes RTP. Tiene rutinas separadas para el manejo de los paquetes recibidos en el lado del cliente que se proporcionan en este trabajo y no es necesario modificarlas. *Se tendrá que completar el primer constructor de esta clase para implementar el empaquetado de RTP de los datos de video.* El segundo constructor es utilizado por el cliente para desempaquetar los datos. No es necesario modificar éste.

VideoStream
Esta clase se utiliza para leer los datos de vídeo desde el archivo en el disco. No es necesario modificar esta clase.

Ejecución del Código

Después de completar el código, puede ejecutarse de la siguiente manera:

En primer lugar, iniciar el servidor con el comando

```
java Server server_port
```

donde `server_port` es el puerto del servidor en el que escucha las conexiones entrantes RTSP. El puerto RTSP estándar es 554, pero se tendrá que elegir un número de puerto superior a 1024.

A continuación, iniciar el cliente con el comando

```
java Client server_name server_port video_file
```

donde `server_name` es el nombre de la máquina donde se ejecuta el servidor, `server_port` es el puerto en el que el servidor está escuchando, y `video_file` es el nombre del archivo que se está solicitando (se proporciona como ejemplo el archivo `movie.mjpeg`). El formato del archivo se describe en el apéndice.
El cliente abre una conexión con el servidor y aparece una ventana como esta:

Pueden enviarse comandos RTSP al servidor pulsando los botones. Una interacción normal RTSP se lleva a cabo de la siguiente manera.

1. El cliente envía SETUP. Este comando se utiliza para configurar los parámetros de la sesión y de transporte.
2. El cliente envía PLAY. Este comando inicia la reproducción.
3. El cliente puede enviar PAUSE si quiere hacer una pausa durante la reproducción.
4. El cliente envía TEARDOWN. Esto termina la sesión y cierra la conexión.

El servidor responde siempre a todos los mensajes que el cliente envía. Los códigos de respuesta son más o menos los mismos que en HTTP. El código 200 significa que la solicitud se ha realizado con éxito. En esta práctica no es necesario implementar algún otro código de respuesta. Para obtener más información acerca de RTSP, consultar el RFC 2326.

1. Cliente

La primera tarea es implementar RTSP en el lado del cliente. Para ello, es necesario completar las funciones que son llamadas cuando el usuario hace clic en los botones de la interfaz de usuario. Para cada botón en la interfaz hay una función controladora (*handler*) en el código. Se tendrá que implementar las siguientes acciones en cada función *handler*.
Cuando el cliente inicia, también se abre el socket RTSP al servidor. Usar este socket para enviar todas las solicitudes RTSP.

SETUP

- Crear un socket para recibir datos RTP y establecer el tiempo de espera (timeout) en el socket a 5 milisegundos.
- Enviar solicitud SETUP al servidor. Se tendrá que insertar el encabezado de Transporte en el que se especifica el puerto para el socket de datos RTP que se acaba de crear.
- Leer la respuesta del servidor y analizar la cabecera de Sesión en la respuesta para obtener el identificador (ID) de sesión.

PLAY

- Enviar solicitud PLAY. Debe insertarse el encabezado de Sesión y utilizar el ID de sesión devuelto en la respuesta de SETUP. No debe ponerse la cabecera de Transporte en esta solicitud.
- Leer la respuesta del servidor.

PAUSE

- Enviar solicitud PAUSE. Debe insertarse el encabezado de sesión y utilizar el ID de sesión devuelto en la respuesta de SETUP. No debe ponerse la cabecera de Transporte en esta solicitud.
- Leer la respuesta del servidor.

TEARDOWN

- Enviar solicitud TEARDOWN. Debe insertarse el encabezado de Sesión y usar el ID de sesión devuelto en la respuesta de SETUP. No

debe ponerse la cabecera de Transporte en esta solicitud
- Leer la respuesta del servidor.

Nota: *Debe insertarse el encabezado CSeq en cada petición que se envía.* El valor de la cabecera CSeq es un número que se incrementa en uno por cada solicitud que envíe.

Ejemplo

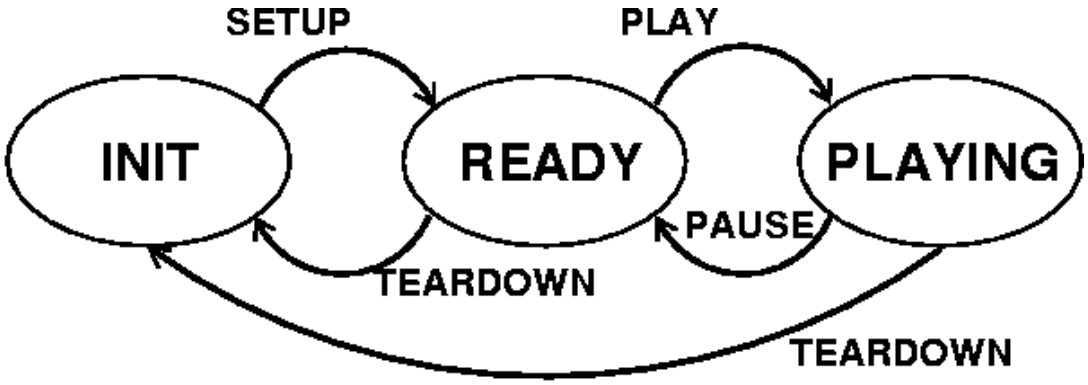
He aquí una muestra de la interacción entre el cliente y el servidor. Las solicitudes del cliente están marcadas con C: y las respuestas del servidor con S:. En esta práctica de laboratorio tanto el cliente como el servidor son **muy simples.** No tienen rutinas de análisis (parsing) sofisticadas y *esperan que los campos de cabecera estén en el orden que aparece a continuación.* Es decir, en una solicitud, la primera cabecera es CSeq, y la segunda es ya sea de Transporte (para SETUP) o Sesión (para todas las otras solicitudes). En la respuesta, CSeq es de nuevo el primero y Sesión el segundo.

```
C: SETUP movie.mjpeg RTSP/1.0
C: CSeq: 1
C: Transport: RTP/UDP; client_port= 25000

S: RTSP/1.0 200 OK
S: CSeq: 1
S: Session: 123456

C: PLAY movie.mjpeg RTSP/1.0
C: CSeq: 2
C: Session: 123456

S: RTSP/1.0 200 OK
S: CSeq: 2
S: Session: 123456

C: PAUSE movie.mjpeg RTSP/1.0
C: CSeq: 3
C: Session: 123456

S: RTSP/1.0 200 OK
S: CSeq: 3
S: Session: 123456

C: PLAY movie.mjpeg RTSP/1.0
C: CSeq: 4
C: Session: 123456

S: RTSP/1.0 200 OK
S: CSeq: 4
S: Session: 123456

C: TEARDOWN movie.mjpeg RTSP/1.0
```

```
C: CSeq: 5
C: Session: 123456

S: RTSP/1.0 200 OK
S: CSeq: 5
S: Session: 123456
```

Estado del cliente

Una de las principales diferencias entre HTTP y RTSP es que en RTSP cada sesión tiene un estado. En esta práctica de laboratorio se necesitará mantener el estado del cliente actualizado. El cliente cambia de estado cuando recibe una respuesta del servidor de acuerdo con el siguiente diagrama de estado.



2. Servidor

En el servidor se tendrá que implementar el empaquetado de los datos de vídeo en paquetes RTP. Para ello, se tendrá que crear el paquete, configurar los campos en la cabecera del paquete y copiar la carga útil (*payload*) (es decir, una trama de vídeo) en el paquete.

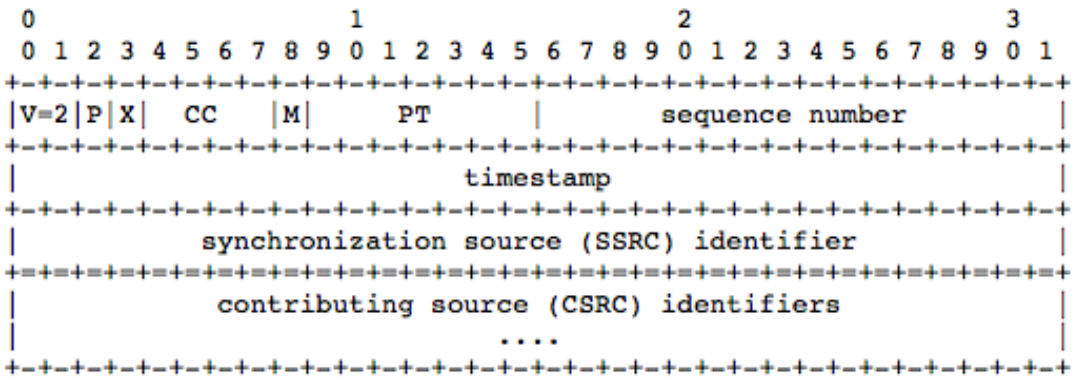
Cuando el servidor recibe la petición PLAY del cliente, se inicia un temporizador que se activa cada 100 ms. En cada uno de estos momentos el servidor leerá una trama de vídeo del archivo y lo enviará al cliente. El servidor crea un objeto RTPpacket que es la encapsulación RTP de la trama de video.

El servidor llama al primer constructor de la clase RTPpacket para llevar a cabo la encapsulación. La tarea será escribir esta función. Se tendrá que hacer lo siguiente: (las letras entre paréntesis se refieren a los campos en el formato de paquetes RTP a continuación)

1. Establecer el campo versión RTP (V). Debe ser 2.
2. Establecer los campos de relleno (*padding*) (P), extensión (X), el número de fuentes que contribuyen (CC), y el marcador (M). *Todos ellos se ponen a cero en esta práctica de laboratorio.*
3. Establecer el campo de tipo de carga útil (PT). En esta práctica se utilizará MJPEG y el tipo de éste es 26.

4. Establecer el número de secuencia. El servidor proporciona este número de secuencia `Framenb` como argumento al constructor.
5. Establecer el timestamp. El servidor proporciona este número como el argumento `Time` al constructor.
6. Establecer el identificador de fuente (SSRC). Este campo identifica al servidor. Puede seleccionarse cualquier valor entero.

Debido a que no tenemos otras fuentes que contribuyen (campo CC == 0), el campo CSRC no existe. Por consiguiente, la longitud de la cabecera del paquete es de 12 bytes, o las tres primeras filas del diagrama de abajo.



Debe llenarse el encabezado en el array `header` de la clase RTPpacket. También se tendrá que copiar la carga útil (dado como argumento `data`) a la variable `payload`. La longitud de la carga útil se proporciona en el argumento `data_length`.

El diagrama de arriba está en el orden de bytes de red (también conocido como big-endian). La máquina virtual de Java utiliza el mismo orden de bytes de modo que no es necesario transformar la cabecera del paquete al orden de bytes de red.

Para más detalles sobre RTP, consultar el RFC 1889.

Desplazando Bits

Estos son algunos ejemplos sobre cómo configurar y comprobar bits individuales o grupos de bits. Notar qué en el formato de cabecera del paquete RTP los bits con números más pequeños se refieren a los bits de orden superior o más significativos, es decir, el bit número 0 de un byte es 2^7 y el número de bit 7 es 1 (o 2^0). En los siguientes ejemplos, los números de bits se refieren a los números en el diagrama de arriba.

Debido a que el campo `header` de la clase RTPpacket es un array de tipo `byte`, se tendrá que establecer la cabecera un byte a la vez, es decir, en grupos de 8 bits. El primer byte tiene los bits 0-7, el segundo byte tiene bits de 8-15, y así sucesivamente. En Java un `int` es de 32 bits o 4 bytes.

Para establecer el número de bit *n* en la variable `mybyte` de tipo byte:

```
mybyte = mybyte | 1 << (7 - n);
```

Para establecer los bits *n* y *n + 1* al valor de `foo` en la variable `mybyte`:

```
mybyte = mybyte | foo << (7 - n);
```

Notar que `foo` debe tener un valor que pueda ser expresado con 2 bits, es decir, 0, 1, 2, o 3.

Para copiar un entero de 16 bits `foo` en 2 bytes, `b1` y `b2`:

```
b1 = foo >> 8;
b2 = foo y 0xFF;
```

Después de esta operación, `b1` tendrá los 8 bits de orden superior de `foo` y `b2` tendrá los 8 bits de orden inferior de `foo`.

Se puede copiar un entero de 32 bits en 4 bytes de una manera similar.

Se puede encontrar más información sobre operaciones de bits en el Tutorial de Java.

Ejemplo de Bit

Supongamos que queremos llenar el primer byte de la cabecera del paquete RTP con los siguientes valores:

- V = 2
- P = 0
- X = 0
- CC = 3

En binario esto se representa como

```
1 0 | 0 | 0 | 0 0 1 1
V=2   P   X   CC = 3

2^7 . . . . . . . . 2^0
```

Apéndice

En esta práctica de laboratorio, el servidor transmite un video que ha sido codificado en un formato de archivo MJPEG propietario. Este formato almacena el vídeo como imágenes concatenadas codificadas en JPEG, con cada imagen precedida por una cabecera de 5 bytes que indica el tamaño en bits de la imagen. El servidor analiza (parsing) el flujo de bits del archivo MJPEG para extraer las imágenes JPEG al vuelo. El servidor envía las imágenes al cliente en intervalos periódicos. El cliente muestra las imágenes JPEG individuales a medida que llegan desde el servidor.