Práctica 4 [Redes 2017]

Integrante(s): Bernal Cedillo Enrique Antonio
Velásquez Garzón Angie Daniela

- **¿Qué pasaría si la red esporádicamente pierde paquetes, cómo afectaría esto al audio y al video?**
  La biblioteca que se está utilizando para la transferencia de información que es *xmlrpc* utiliza como protocolo de transferencia a HTTP el cual corre sobre TCP.
  El protocolo TCP garantiza que los datos son entregados y en el orden que se transmitieron así retransmitirá los paquetes que no hayan llegado correctamente al destinario, para este caso el audio y el video se verían afectados respecto al retraso con el cual los datos serían recibidos por el usuario receptor, donde mientras el usuario receptor está escuchando algo, el emisor ya habrá emitido dicha información en un lapso de tiempo anterior. Lo cual dificultará eventualmente la comunicación entre los usuarios.
  El hecho de que los paquetes se retransmitan ocasiona que no exista sintonía entre la comunicación de los usuarios y sea difícil comunicarse. Dado que la información se almacena en una cola para ambos casos hasta que no hayan llegado todos los paquetes para transmitirlos la comunicación se detendrá y una vez esta se reanude se habrán desincronizado los tiempos porque mientas los paquetes se enviaban al usuario receptor el usuario emisor continuo con la transferencia de paquetes.

- **Respecto a la pregunta anterior, ¿cómo afecta esta situación al desempeño de la red?**
  La retransmisión de paquetes generará un alto tráfico de información (paquetes) en la red conllevando a una congestión de la red, pues por cada paquete que no se pueda entregar correctamente se deberá realizar un esfuerzo por enviarlo nuevamente, además de que se deberá continuar con la transmisión de los nuevos paquetes que se generaron durante la transmisión del audio y el video.

- **¿Qué solución puede darse para evitar los problemas de las preguntas anteriores, y en qué capa del modelo osi puede resolverse?**
  Una posible solución consiste en permitir la pérdida de paquetes lo cual evitaría tener que re-enviar paquetes y asi congestionar la red. Esto haría que el video diera “saltos” esporádicamente y tampoco aseguraría la sincronización de la comunicación. [Usar UDP]
  Otra solución consiste en utilizar multiples conexiones TCP de forma paralela para el envio de paquetes, por lo que el retraso y re-envio de un paquete no retrasaría el envio de los paquetes subsecuentes, sin embargo es poco recomendable por el costo de mantener las conexiones. (Dichos cambios deberían modificarse en la capa de transporte y red)

  Una alternativa más se basa en limitar la resolución, framerate y gama de la captura de la cámara para asi minimizar el espacio requerido por las imágenes a transmitir y gastar menor tiempo en el envío de dichas imágenes.
  Por lo que se requieren menos paquetes para mantener el envio del video, al costo de reducir la calidad de la imagen. (Se realizaría en la capa de aplicación)

  (Idea general parafraseada de: http://wind.lcs.mit.edu/papers/pv2002.pdf)

Reporte del código

- **Flujo del programa**

1. Inicia la GUI del *Login* la cual de acuerdo al modo en que se inició la sesión, será local o remota, pedirá al usuario:
    - **Local:** Número del puerto por el cual desea establecer su conexión y el número del puerto al que desea conectarse.
    - **Remoto:** Dirección IP del equipo con el cual desea conectarse

2. El usuario ingresa los datos solicitados, el canal se encarga de iniciar los hilos para las instancias del cliente y servidor de cada usuario y se establece una conexión si la información ingresada es válida, de lo contrario se informa al usuario que la conexión no se ha podido establecer, paso seguido se presenta la pantalla del chat donde el usuario puede intercambiar mensajes de texto o audio.

3. El usuario tendrá disponibles dos modalidades para la comunicación, a través de mensajes de texto o de audio:

- **Mensajes:** Si el usuario escribe un mensaje y presiona el botón *enviar* el canal a través de su cliente establece una conexión al contacto indicado cuando inició sesión y envía el mensaje al servidor del usuario receptor y este se encarga de mostrarlo en la ventana de dialogo.

- **Audio:** Si el usuario presiona el botón de *llamar* el canal iniciará dos hilos, uno que se encargara de ir grabando el audio e ir encolando la información en una cola y el otro que se encargará de ir tomando lo que se encuentra en la cola y enviarlo al servidor del contacto el cual se encargará de reproducirlo, esté procedimiento se llevará a cabo todo el tiempo que la llamada se encuentre activa y no se permitirá el envío de mensajes.

- **Video:** Cuando se presiona el botón de videollamada, se crearán los hilos correspondientes para el envio y reproducción de secuencia de imágenes (de manera análoga al envio de audio).

- **Carpeta GUI**

    - **ChatGUI.py**
      Clase que a través de una interfaz gráfica le permite al usuario hacer uso de las funcionalidades de comunicación: *envío de mensajes* y *llamadas.* Está clase contiene el canal quien se encarga de establecer la comunicación, enviar y recibir información entre los usuarios del chat.
    - **LoginGUI.py**
      Contiene clase LoginGUI que se encarga de solicitar la información correspondiente al usuario para establecer la conexión entre los usuarios. La información solicitada varía de acuerdo al modo en que se indique por consola si la conexión es local o remota.

- **Carpeta Channel**
  - **ApiClient.py**
    Clase encargada de representar al cliente de la conexión, permite establecer la comunicación con el servidor del usuario contacto para el envío de mensajes o inicio de una llamada.
    Cuando se desea enviar un mensaje, se establece la conexión con el servidor del contacto y se envía el mensaje.
    Cuando se inicia una llamada, el cliente inicia una instancia de un cliente audio e inicia un hilo que permite grabar de forma continua el audio y guardar dicha información en una cola mientras de forma paralela el cliente establece comunicación con el servidor del usuario de contacto y repetitivamente extrae la información de la cola y la envía al servidor de contacto.

  - **ApiServer.py**
    Clase encargada de recibir los mensajes y reproducir los audios recibidos por parte del cliente del contacto y disponérselos a su usuario. El servidor cuenta con dos servicios, uno que permite recibir los mensajes y mostrarlos en la pantalla de su usuario y otro que se encarga de reproducir el audio capturado por el cliente del contacto.

  - **RecordAudio**
    Archivo que contiene un cliente y servidor para la transferencia de audio entre dos usuarios, la clase de *AudioClient* cuenta con los servicios necesarios para la grabación del audio de su usuario; la clase *AudioServer* provee el servicio necesario para reproducir la información que recibe.

  - **RecordVideo**
    Archivo que contiene un cliente y servidor para la transferencia de video entre dos usuarios, la clase de *VideoClient* cuenta con los servicios necesarios para la grabación del audio de su usuario; la clase *VideoServer* provee el servicio necesario para mostrar la información que recibe.
    Todo esto, apoyado por openCV.

  - **Channel.py**
    Alberga la clase *Channel* la cual administra los hilos del cliente y servidor de cada usuario, gestiona el envío de mensajes, llamadas (para la cual inicia un nuevo hilo del cliente que se encarga de enviar continuamente al servidor de contacto el audio) y finalización de llamadas. Maneja los hilos para iniciar el cliente y servidor, además de gestionar la llamada.

- **Principales problemas encontrados y cómo se solucionaron**
  No se conocía el funcionamiento de la biblioteca OpenCV, lo cual dificultó un poco el inicio de avances con la práctica.

- **Problemas no solucionados**
  Detener la transmisión de imágenes de la cámara provoca errores.