Práctica 3 - Multimedia

Sitio:	MOODLE DE GRADO
Curso:	REDES DE COMUNICACIONES II
Libro:	Práctica 3 - Multimedia
Imprimido por:	Javier Delgado del Cerro
Día:	martes, 18 de junio de 2019, 19:51

Tabla de contenidos

1 Introducción

En esta última práctica, abordaremos el área de la transmisión multimedia. Nuestro objetivo es simple: *crear una pequeña aplicación que nos permita transmitir video, sin audio, entre dos clientes*, utilizando un protocolo P2P propio. Para ello, se ha diseñado un protocolo sencillo pero funcional que cumple, al menos, los siguientes requisitos:

- Solo es necesario que soporte comunicación unicast entre dos únicos pares simultáneos (es decir, no es multiconferencia).
- El cliente debe permitir tanto la transmisión en tiempo real de imágenes desde una webcam o la transmisión de un fichero de video.
- Debe soportar las operaciones básicas de PLAY, PAUSE y STOP del flujo multimedia.
- Control de flujo mínimo según la especificación del enunciado.

La versión básica (0) es la funcionalidad mínima pedida y deben ser soportadas por todas las entregas para que sean compatibles entre sí.

A partir de la funcionalidad mínima pedida en la versión 0 del protocolo, se pueden incorporar mejoras como:

- Refinar cuestiones como la posible congestión de la red, los diferentes ritmos a los que cada parte puede consumir o generar el flujo, etc.
- Soporte para audio.
- Multiconferencia.
- Otras mejoras (siempre relacionadas con redes).

Cuando se incorporen las mejoras, **si cambian la compatibilidad con la versión 0 es necesario definir un nuevo código de versión**, pero el programa siempre debe soportar también la versión 0.

2 Arquitectura y servidor de descubrimiento

A grandes rasgos, la aplicación P2P de teleconferencia deberá permitir que el usuario realice las siguientes acciones:

- **Gestionar sus nicks (creación y cambio)**. Para ello deberá solicitar un nick al usuario, y registrarlo a continuación en el **servidor de descubrimiento**. Este nick tendrá una validez de 24 horas, periodo durante el que estará "reservado", a través del uso de una contraseña, a dicho usuario. Pasado ese tiempo, el servidor eliminará el nick, y otro usuario podrá reclamarlo y registrarlo para sí.
- **Realizar una videollamada con otro usuario**. Para ello la aplicación deberá permitir que el usuario escriba (o elija de una lista) el nick del usuario con el que desea conectar. Tras ello, abrir una conexión directa con este usuario (tal y como se explica en las siguientes secciones), y empezar a capturar, enviar y recibir el video. También deberá permitir pausar y finalizar la llamada.

Servidor de descubrimiento (DS)

Como sabemos, los servicios de videoconferencia, como Skype o el que desarrollamos en esta práctica, suelen seguir el esquema P2P, con la conexión directa entre los nodos participantes en la comunicación. Sin embargo, este sistema necesita de un nodo especial, fijo, que funcione a modo de tablón de anuncios donde cada nuevo participante en la red pueda "colgar" su dirección IP y puerto de escucha, de forma que el resto de participantes puedan conectarse a él.

En nuestra práctica, este papel lo realiza el **servidor de descubrimiento**, que consiste en un pequeño servicio que escucha en el puerto 8000 del host *vega.ii.uam.es*, y recibe y envía los mensajes como simples cadenas de texto ASCII.

Su función esencial es mantener una pequeña base de datos con todos los usuarios registrados en la plataforma, guardando para cada uno de ellos, *nick, dirección IP, puerto de escucha, un timestamp del momento en el que se registró o actualizó la validez del nick por último vez y protocolos soportados.*

Para ello, soporta los siguientes mensajes:

Función REGISTER
Objetivo: Registrar a un usuario en el sistema. También sirve para cambiar el nick de un usuario existente.
Comando: REGISTER nick ip_address port password protocol
Posibles respuestas/errores:

- OK WELCOME nick ts : El registro o actualización del nick se ha realizado correctamente. ts contiene el nuevo instante de registro (en formato UNIX)
- NOK WRONG_PASS : El nick es válido, pero la contraseña proporcionada es errónea

```
Ejemplo
C->S: REGISTER mi_nick 138.100.10.100 8080 secret V1#V2
S->C: OK WELCOME mi_nick 1521798459.6474934
Con este mensaje, el cliente registra el nick 'mi_nick', con la dirección IP 138.100.10.100
y puerto 8080 en el servidor. También le asigna la contraseña 'secret' e indica que soporta
los protocolos V1 y V2.
```

Como se ha comentado, una vez registrado por primera vez un nick, el servidor lo mantendrá reservado durante 24h, y los intentos de registrar ese nick por parte de otros usuarios serán rechazados (con una respuesta NOK WRONG_PASS). Para seguir utilizando su nick, el usuario legítimo debe volver a llamar a REGISTER con la contraseña adecuada, momento en el que se le extenderá la validez del mismo otras 24h. Típicamente, la aplicación debería llamar a REGISTER cada vez que sea abierta por el usuario.

Función QUERY
Objetivo: Permite obtener la dirección IP y puerto de un usuario conociendo su nick
Comando: QUERY name
Posibles respuestas/errores:

- OK USER_FOUND nick ip_address port protocols : Devuelve información sobre el usuario. protocols contiene todos los protocolos soportados por ese cliente, separados por '#'.
- NOK USER_UNKNOWN

Función LIST_USERS
Objetivo: Listar todos los usuarios registrados en el sistema
Comando: LIST_USERS
Posibles respuestas/errores:

- OK USERS_LIST user1#user2#...#usern# : Devuelve una lista de los usuarios registrados en el sistema, mostrando para cada uno la información devuelta por el comando QUERY, separados por el símbolo '#'
- NOK USER_UNKNOWN

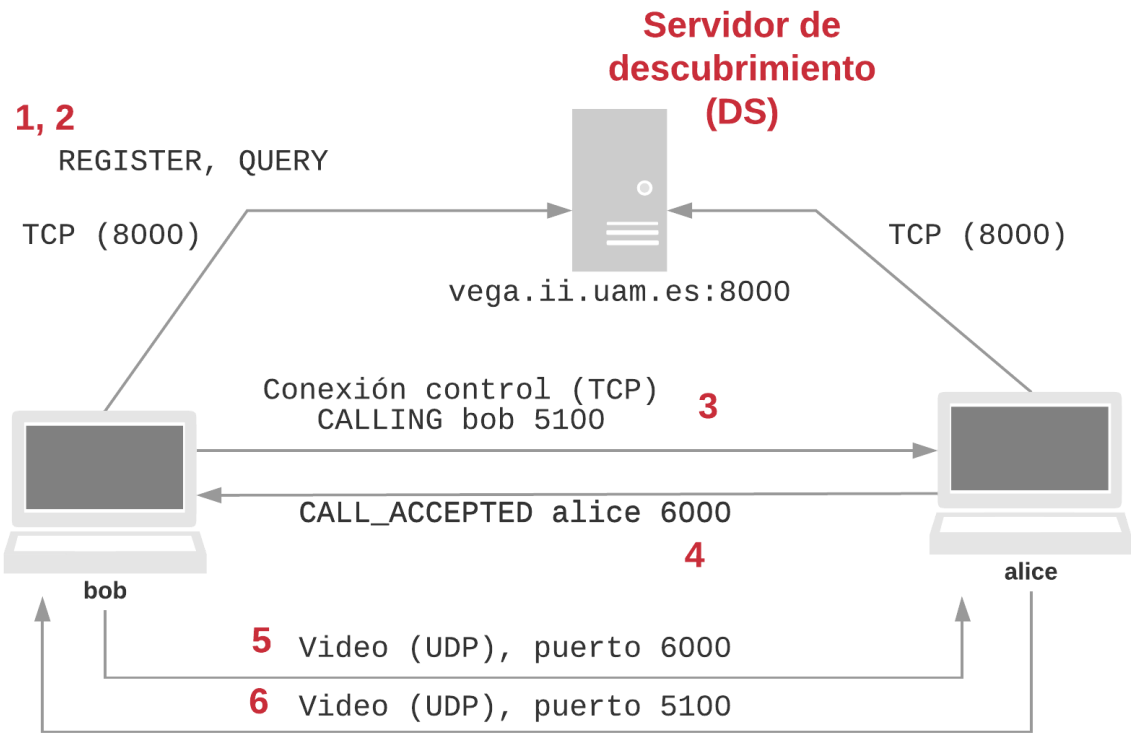
Función QUIT
Objetivo: Señalizar el cierre de la conexión con el servidor
Comando: QUIT
Posibles respuestas/errores:

- BYE

3 Comunicaciones

Arquitectura

A grandes rasgos, la arquitectura del sistema es la que se muestra en la siguiente figura:



En ella se aprecia que hay dos tipos básicos de comunicaciones:

- Conexiones de control TCP con el DS y el nodo destino (alice)
- Datagramas UDP de datos entre los pares (video)

Para realizar una llamada, una secuencia tipo deberá ser similar a la siguiente (mensajes en rojo en la figura):

- **Mensaje 1**: La aplicación registra (o actualiza) el nick del usario en el DS cada vez que ésta se lanza, o el usuario solicitad un cambio de nick, respectivamente. Posteriormente, la aplicación deberá permitir al usuario iniciar una videollamada. Para ello:
  - El usuario podrá escribir directamente el nick deseado, si ya lo conoce.
  - La aplicación mostrará una lista de nicks registrados (obtenidos a través de LIST_USERS) y permitirá al usuario seleccionar uno.
- **Mensaje 2**: La aplicación pide al DS los datos necesarios del nick elegido: dirección IP, puerto y protocolos soportados. El protocolo a usar será el protocolo de máximo nivel común a los dos clientes.
- **Mensaje 3**: La aplicación inicia la conexión de control con el cliente, en la dirección IP y puerto especificados en el mensaje anterior.
- **Mensaje 4**: La otra parte aceptará o declinará la llamada con CALL_ACCEPTED o CALL_DENIED, respectivamente.
- **Mensajes 5 y 6**: Los datos de video se transmiten por UDP al puerto recogido en el comando CALL_ACCEPTED.

Obviamente, la aplicación debe ser capaz de hacer todo esto simultáneamente, atendiendo al interfaz gráfico, recibiendo y emitiendo video, y atendiendo a los comandos de control a la vez con diferentes hilos.

Comandos de control

Los comandos de control son los mensajes que se intercambian directamente los pares para controlar todos los aspectos de la videollamada, como señalización de inicio, pausa, final, etc. Los especificados en esta sección son los mínimos a implementar. Cada pareja es libre de diseñar e implementar más, siempre respetando las sintaxis de éstos, para que todos los clientes sean compatibles entre sí (más sobre esto en la sección de protocolos).

Comando CALLING

Objetivo: Señalizar que un nodo quiere establecer una videollamada con otro
Sintaxis: CALLING nick srcUDPport, donde *srcUDPport* es el puerto UDP en el que el llamante desea recibir el video del llamado.
Posibles respuestas/errores:

- CALL_ACCEPTED nick dstUDPport, donde dstUDPport es donde el llamado recibirá el video
- CALL_DENIED nick
- CALL_BUSY, el llamado está en una llamada y no puede recibir la comunicación.

Comando CALL_HOLD
Objetivo: Señalizar que se desea pausar temporalmente una llamada, sin cortarla.
Sintaxis: CALL_HOLD nick
Posibles respuestas/errores: Ninguno

Comando CALL_RESUME
Objetivo: Señalizar que se desea reanudar una llamada anteriormente pausada.
Sintaxis: CALL_RESUME nick
Posibles respuestas/errores: Ninguno

Comando CALL_END
Objetivo: Señalizar que se desea finalizar una llamada.
Sintaxis: CALL_END nick
Posibles respuestas/errores: Ninguno

Gestión del flujo de video

El control de flujo de video es también una parte importante de la práctica, que es donde más libertad se proporcionará. En principio, el objetivo es ser capaz de gestionar situaciones de congestión y diferentes anchos de banda entre los participantes, adaptantado el flujo de video para que la calidad de la videollamada sea la mejor posibles en cada momento. Para ello, se podrá jugar esencialmente con dos parámetros:

- Calidad del video emitido
- Frames por segundo

Ambos parámetros pueden modificarse a través de las funciones adecuadas, que vienen comentadas en el código fuente de partida entregado. Para poder detectar y gestionar una posible congestión, el formato de los paquetes de datos UDP incluirá una cabecera y tendrá el siguiente formato:

- Número de orden#Timestamp#Resolución video#FPS#Datos...

donde:

- *Número de orden* es un identificador del paquete, que la fuente del video incrementa en cada frame, y que servirá al receptor para ordenar, si es necesario, los paquetes desordenados.
- *Timestamp*, en formato UNIX, del momento en el que se envió el paquete desde el origen. Esto permitirá detectar el posible retraso y, si es superior a cierto límite, poner en marcha las medidas anti-congestión.
- *Resolución* del video emitido, en formato simple de cadena. Por ejemplo, "640x480".
- *FPS*: frames per second.
- *Datos*: frame a visualizar en el receptor. Los datos deben ir **comprimidos** para que cada frame pueda caber en un solo datagrama UDP. Para ello usaremos un **códec** muy sencillo en el que cada frame se comprime como una imagen JPEG. Proporcionamos un ejemplo:

```
ret, img = self.cap.read() # lectura de un frame de vídeo

# Compresión JPG al 50% de resolución (se puede variar)
encode_param = [cv2.IMWRITE_JPEG_QUALITY,50]
result,encimg = cv2.imencode('.jpg',img,encode_param)
if result == False: print('Error al codificar imagen')
encimg = encimg.tobytes()

# Los datos "encimg" ya están listos para su envío por la red
#enviar(encimg)

# Descompresión de los datos, una vez recibidos
decimg = cv2.imdecode(np.frombuffer(encimg,np.uint8), 1)

# Conversión de formato para su uso en el GUI
cv2_im = cv2.cvtColor(decimg,cv2.COLOR_BGR2RGB)
img_tk = ImageTk.PhotoImage(Image.fromarray(cv2_im))
... etc
```

Normalmente, este tipo de clientes se codifican utilizando *buffers circulares*, y sería también la solución más adecuada aquí. De esta forma, el receptor del video no comenzará a visualizar el video inmediatamente, sino que lo hará cuando se haya llenado este búfer, y así tenga un margen de seguridad en caso de pequeños cortes. El tamaño del búfer (número de slots) dependerá de cada cliente, pero una regla sencilla para calcularlo puede ser: queremos almacenar 2 segundos de video de margen, a 20 fps = 40 slots.

Protocolos

Para permitir que aquellos estudiantes que así lo deseen puedan diferenciar su práctica añadiéndole mejoras y características extra, el protocolo básico soportará el versionado del mismo.

Así, el protocolo básico descrito hasta el momento tendrá el número de versión V0, que deberá especificarse en el comando REGISTER, y deberá ser soportado por todos los clientes. Si alguna pareja desea añadir nuevas funcionalidades, como un algoritmo mejorado de control de flujo, múltiples flujos de video, cifrado de las comunicaciones, etc., deberá:

1. Escribir una pequeña descripción de la funcionalidad de su protocolo, sintaxis de los mensajes, etc.
2. Contactará y consensuará con su profesor de prácticas la versión definitiva. Una vez hecho esto, se publicará en esta sección para que el resto de estudiantes que así lo quieran puedan también implementarlo. Obviamente, la implementación de estos protocolos extra hará que la puntuación final de la práctica sea superior.

4 Recursos

Para llevar a cabo la práctica se proporcionan una serie de recursos:

- La conexión inicial entre pares P2P necesita siempre un servidor de descubrimiento donde consultar su dirección IP y puerto. Para ello, se ha creado un servidor específico, que se describe en la siguiente sección, disponible en el host *vega.ii.uam.es*, puerto 8000.
- El esqueleto del cliente P2P se proporciona también hecho. En función de la funcionalidad que se implemente, puede ser necesario modificar el GUI existente, que está basado en este framework.
- Para la captura de imágenes de la webcam, o la transmisión de un fichero de video, el binding para Python de la librería OpenCV puede ser muy útil. A buen entendedor, pocas palabras bastan...

Cliente P2P

Como se ha comentado, el esqueleto del cliente P2P se proporciona hecho, en el repositorio Git de la asignatura. Si lo ejecutas en tu propia máquina, necesitarás instalar una serie de paquetes:

```
#sudo apt install python3-pil.imagetk python3-tk python3-numpy python-imaging-tk
```

Para OpenCV debe servir con **`sudo apt install python-opencv`** , aunque quizás sea necesario **`pip3 install opencv-python`**

Si en la máquina hay conectada una webcam, debería funcionar sin necesidad de configuración adicional con el siguiente comando:

```
# python3 video_client.py
```

Estudia el código para localizar dónde debes incluir la funcionalidad requerida.

Visualización de ambos vídeos simultáneamente

Ejemplo para superponer dos imágenes con OpenCV, una más pequeña (frame_top) dentro de la otra (frame_base)

```
frame_peque = cv2.resize(frame_top, (320,240)) # ajustar tamaño de la imagen pequeña
frame_compuesto = frame_base
frame_compuesto[0:frame_peque.shape[0], 0:frame_peque.shape[1]] = frame_peque
```

También se puede utilizar una ventana adicional que muestre el otro vídeo durante la llamada.

Esto son sugerencias, se aceptará cualquier solución que permita ver a la vez los dos vídeos (el emitido y el recibido).

Empaquetado de vídeo para la transmisión por red

Como se ha indicado, es necesario incorporar una cabecera a los datos de vídeo comprimidos en JPG, previamente a transmitir los frames por la red.

La cabecera es en texto plano y debe ir concatenada antes de los datos JPG en bytes.

Esto se puede conseguir de varias formas:

- Definiendo las cadenas como bytes poniendo "b" delante: b'cadena en bytes'
- Con la función bytearray() para hacer casting de strings a bytes.
- Con las funciones "struct.pack" y "struct.unpack" incluidas en la librería "struct" de python.
- Mediante otros métodos que se os ocurran y no afecten a la compatibilidad entre prácticas de otros grupos.