

Protocolos para la Transmisión de Audio y Vídeo por Internet

Práctica Final Parte Obligatoria (Versión v11.0, 16.12.2020)

December 16, 2020

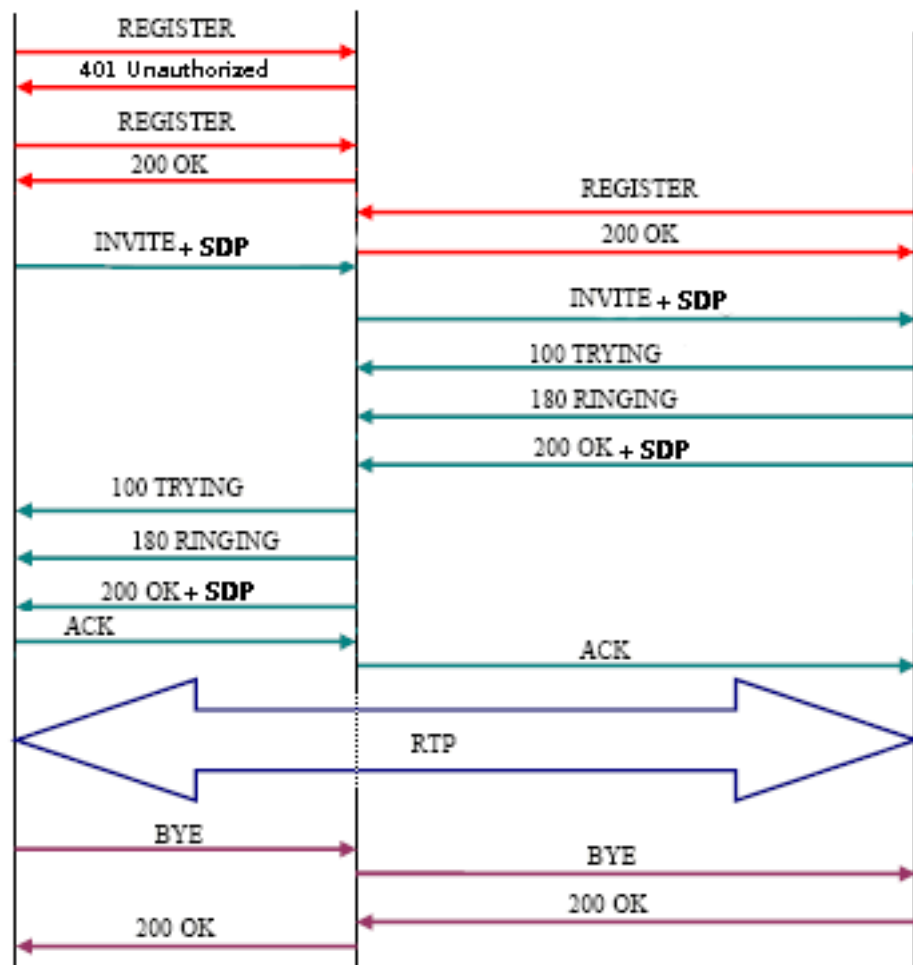
Contents

1	Introducción	3
1.1	Conocimientos previos necesarios	4
1.2	Entrega	4
2	User Agent	5
2.1	Configuración	5
2.2	Log	6
2.3	User Agent Client (UAC)	6
2.3.1	Métodos	7
2.4	User Agent Server (UAS)	8
2.4.1	Códigos de respuesta	9
2.4.2	Envío RTP	9
3	Servidor Proxy/Registrar	10
3.1	Configuración	10
3.2	Códigos de respuesta	11
3.3	Log	11
3.4	Base de datos de usuarios registrados	11
4	Ejercicios alternativos	11
4.1	Ejercicio A	12
4.2	Ejercicio B	12
4.3	Ejercicio C	13
4.4	Ejercicio D	13

4.5	Ejercicio E	14
4.6	Ejercicio F	14
4.7	Ejercicio G	14
4.8	Ejercicio H	15
5	Capturas	15
5.1	Captura llamada.libpcap	16
5.2	Captura error.libpcap	16
6	Preguntas frecuentes	17

1 Introducción

Esta práctica tiene como objetivo implementar los programas en Python necesarios para realizar una sesión SIP de intercambio de audio vía RTP como la que se muestra en la siguiente figura¹:



En esta sesión:

¹Las respuestas SIP aparecen en mayúsculas en la figura, por claridad. En la práctica, siguiendo el RFC, sólo la primera letra de cada palabra ha de ser mayúscula.

- El UA de la izquierda constará de una parte cliente y otra servidora (se puede partir de los programas realizados en la práctica 6).
- El UA de la derecha constará de una parte cliente y otra servidora (se puede partir de los programas realizados en la práctica 6).
- Habrá un servidor de registro (se puede partir del programa realizado en la práctica 4) y de *proxy*, que intermediará únicamente en el intercambio SIP; los paquetes RTP irán directamente de UA emisor a UA receptor, sin pasar por el proxy/registro.

1.1 Conocimientos previos necesarios

- Conocimientos del protocolo SIP (las de clase de teoría)
- Funcionamiento de Wireshark

Tiempo estimado: 25 horas

1.2 Entrega

Con el navegador, dirígete al repositorio `ptavi-pfinal` en la cuenta del profesor en GitLab² y realiza un `fork`, de manera que consigas tener una copia del repositorio en tu cuenta de GitLab. Clona el repositorio que acabas de crear a local para poder editar los archivos. Trabaja a partir de ahora en ese repositorio, sincronizando los cambios que vayas realizando.

La entrega de la parte obligatoria de la práctica final se deberá hacer antes del **miércoles 3.2.2021 a las 15:00**. Para entonces, se debe tener un repositorio git en GitLab con los siguientes archivos:

1. `uaclient.py` (cliente del UA)
2. `uaserver.py` (servidor del UA)
3. `proxy_registro.py` (servidor que hace funciones de `proxy` y `Registrar`)
4. `ua1.xml` (fichero de configuración del UA)
5. `ua2.xml` (fichero de configuración del UA)
6. `pr.xml` (fichero de configuración del `proxy/registro`)
7. `llamada.libpcap` (captura de una llamada exitosa)
8. `error.libpcap` (captura de una llamada con error)

²<http://gitlab.etsit.urjc.es/ptavi/ptavi-pfinal>

9. `passwords.json` (fichero con los usuarios y sus contraseñas guardados en el proxy-registrar)

10. Otros: `README.md`, `.gitignore`, `.git`, `check-pfinal.py`, `cancion.mp3`

Otras consideraciones:

- Se tendrá en cuenta para la nota de la práctica la adecuación de los programas Python a las guías de estilo de Python PEP8 y PEP257.
- Se valorará el uso de git (frecuencia de commits y adecuación de los comentarios de cada commit).
- Se valorará que la práctica siga las convenciones indicadas en este documento, en particular que el intercambio de mensajes, los parámetros de los programas y los mensajes en el fichero de *log* sean los que especifican. En la *shell* pueden incluirse las trazas que el alumno desee y en el formato que le parezca más conveniente.
- Se valorará que las capturas estén bien filtradas y sólo contengan los paquetes que se indican en el guión.
- Se valorará que el cliente y el servidor siga el estándar SIP (o en su caso, las simplificaciones propias de esta práctica) al pie de la letra.
- Se proporciona un script de comprobación de entrega correcta, *check-pfinal.py*. Es responsabilidad del alumno comprobar la correcta entrega mediante este script.
- Se pasará un programa anti-copia a las prácticas entregadas. El alumno es responsable de su código, y debe poder saber explicarlo y defenderlo.

2 User Agent

Cada *User Agent* deberá constar de una parte cliente y otra servidora, que podrán estar ubicados en programas Python diferentes (y, por tanto, requerirán de dos terminales para ser ejecutados).

2.1 Configuración

La configuración del *User Agent* estará almacenada en un fichero con formato XML válido, que seguirá la sintaxis que se describe en un DTD³ que viene dado en los ejercicios alternativos A y B, y donde los elementos son los siguientes:

- `account`: Cuenta del usuario, con su nombre y su contraseña.

³Se puede profundizar en la sintaxis de los DTD en: <http://www.mundolinux.info/sintaxis-de-las-dtd.htm>

- `uaserver`: Lugar de escucha de la parte servidora del *User Agent*, indicando su dirección IP (opcional, en caso de no estar indicada se utilizará 127.0.0.1) y el puerto.
- `rtpaudio`: Puerto de recepción para el audio RTP.
- `regproxy`: Parámetros de localización del servidor Proxy/Registrar: IP y puerto.
- `log`: Localización (*path*) del fichero de log del *User Agent*.
- `audio`: Localización (*path*) del fichero MP3.

2.2 Log

El cliente ha de escribir mensajes de depuración y de *log* en un fichero indicado en el fichero XML de configuración. Cada línea de este fichero de log ha de tener un mensaje con el que se pueda trazar qué es lo que ha ocurrido (o está ocurriendo) en el *User Agent*.

Los ficheros de log deben tener la estructura dada en los ejercicios alternativos C y D, donde evento puede ser:

- En caso de envío: *Sent to IP:port: mensaje completo en una única línea*. Los saltos de línea han de transformarse en espacios en blanco.
- En caso de recepción: *Received from IP:port: mensaje completo en una única línea*. Los saltos de línea han de transformarse en espacios en blanco.
- En caso de error (p.ej. porque se ha enviado algo a un puerto que no está escuchando): *Error: mensaje de error en una única línea*. Los saltos de línea han de transformarse en espacios en blanco.
- Otros eventos: Tipo de evento, tal como *Starting...*, *Finishing...*, etc.

2.3 User Agent Client (UAC)

El cliente ha de ejecutarse de la siguiente manera:

```
$ python uaclient.py config metodo opcion
```

donde *config* será un fichero XML con la configuración, *emphmetodo* será un método SIP, y *opcion* será un parámetro opcional dependiendo del método utilizado.

Por ejemplo:

```
$ python uaclient.py ua1.xml REGISTER 3600
$ python uaclient.py ua1.xml INVITE penny@girlnextdoor.com
$ python uaclient.py ua1.xml BYE penny@girlnextdoor.com
$ python uaclient.py ua1.xml REGISTER 0
```

En caso de no introducir el número de parámetros correctos o de error en los mismos, el programa debería imprimir:

```
Usage: python uaclient.py config method option
```

En caso de que se intentara establecer una conexión con un puerto no abierto (p.ej. porque el servidor no haya sido lanzado), se capturará la excepción y, antes de finalizar el programa, se incluirá en el log un texto como el siguiente:

```
20101018160243 Error: No server listening at 193.147.73 port 5555
```

El cliente podrá imprimirá por pantalla trazas, si así se desea. Se recuerda que los mensajes que envíe y reciba de sus comunicaciones han de aparecer en el archivo de *log* con el formato especificado.

2.3.1 Métodos

- REGISTER sip:emisor:puerto SIP/2.0

Con este método el *User Agent* se registra en un servidor de registro. Debido a las circunstancias donde se van a realizar las prácticas -donde los *User Agents* van a estar ejecutándose en la misma máquina- debemos especificar el puerto de escucha del *User Agent* en la petición.

En las cabeceras del REGISTER ha de incluirse siempre una cabecera con el tiempo de expiración.

```
REGISTER sip:leonard@bigbang.org:1234 SIP/2.0
Expires: 3600
```

- INVITE sip:receptor SIP/2.0

Mediante este método, se indica que queremos iniciar una conversación con el receptor con dirección receptor en la máquina dada por la IP (que puede ser 127.0.0.1).

El cuerpo del INVITE debe incluir la descripción de sesión en formato SDP (*Session Description Protocol*), por lo que habrá que añadir las cabeceras correspondiente al INVITE (Content-Type: application/sdp y Content-Length con la longitud correspondiente del cuerpo). Contará con los siguientes parámetros:

- v (versión, por defecto la “0”),
- o (originador e identificador de sesión: la dirección del originador y su IP),
- s (nombre de la sesión, que puede ser el que se desee),
- t (tiempo que la sesión lleva activa, en nuestro caso, siempre 0), y

- m (tipo de elemento multimedia y puerto de escucha y protocolo de transporte utilizados, en esta práctica “audio”, el número de puerto pasado al programa principal como parámetro y “RTP”).

Así, un ejemplo de descripción de sesión dentro de un INVITE podría ser:

```
INVITE sip:penny@girlnextdoor.com SIP/2.0
Content-Type: application/sdp
Content-Length: 76

v=0
o=leonard@bigbang.org 127.0.0.1
s=misesion
t=0
m=audio 34543 RTP
```

Nótese que en el ejemplo anterior 34543 es el puerto donde esperamos que el otro participante en la conversación nos envíe los paquetes RTP con audio. También se ha de tener en cuenta que entre las cabeceras (en este caso la línea de INVITE) y el cuerpo (la descripción de la sesión) ha de haber obligatoriamente una línea en blanco.

- ACK sip:receptor SIP/2.0
Método de asentimiento. No se pasará al programa uaclient.py como parámetro, ya que se enviará una vez se hayan recibido las respuestas *100 Trying*, *180 Ringing* y *200 OK* de la parte servidora tras hacerle un INVITE.
- BYE sip:receptor SIP/2.0
Mediante este método se indica que queremos terminar la conversación que estamos manteniendo con la otra parte. Se deberá llamar, una vez haya acabado el streaming de audio vía RTP. Puede ser llamada desde cualquiera de los dos *User Agents* involucrados en la conversación.

2.4 User Agent Server (UAS)

El servidor ha de ejecutarse de la siguiente manera:

```
$ python uaserver.py config
```

donde *config* será un fichero XML con los parámetros de configuración.

Por ejemplo:

```
$ python uaserver.py ua1.xml
```

En caso de no ejecutarse correctamente, el programa debería imprimir:


```
Usage: python uaserver.py config
```

En caso contrario, el servidor imprimirá por pantalla:

```
Listening...
```

El servidor podrá imprimir trazas por pantalla. Se recuerda que los mensajes que envíe y reciba de sus comunicaciones han de aparecer en el archivo de *log* en el formato especificado con anterioridad.

El servidor guardará la dirección de origen y la dirección IP recibida por SDP en un diccionario⁴ para utilizarlo posteriormente en el envío de RTP.

2.4.1 Códigos de respuesta

Los códigos de respuesta que el UA maneja han de ser, al menos, los siguientes:

- SIP/2.0 100 Trying: al recibir un INVITE. Puede enviarse en un mismo paquete con las respuestas 180 y 200; en ese caso, no hace falta dejar una línea en blanco después del “100 Trying”.
- SIP/2.0 180 Ringing: al recibir un INVITE. Puede enviarse en un mismo paquete con las respuestas 100 y 200; en ese caso, no hace falta dejar una línea en blanco después del “180 Ringing”.
- SIP/2.0 200 OK: Deberá incluir SDP en el cuerpo (con su correspondiente cabecera de Content-Type y Content-Length) si es en respuesta a un INVITE, pero no cuando sea a un BYE.
- SIP/2.0 400 Bad Request: si la petición está mal formada.
- SIP/2.0 404 User Not Found: cuando el usuario no se ha encontrado en el servidor de registro.
- SIP/2.0 405 Method Not Allowed: si se manda en la petición cualquier otro método diferente de INVITE, BYE o ACK.

2.4.2 Envío RTP

El envío RTP se realizará mediante la biblioteca `simplertp`⁵. Un ejemplo del código Python de envío de paquetes RTP sería el siguiente:

```
import simplertp
```

```
RTP_header = simplertp.RtpHeader()
```

⁴Este diccionario ha de ser un atributo de clase no de la instancia, véase <http://www.toptal.com/python/python-class-attributes-an-overly-thorough-guide>

⁵En el repositorio se encuentra el módulo Python `simplertp.py`

```
RTP_header.set_header(marker=BIT, ssrc=ALEAT)
audio = simplertp.RtpPayloadMp3(audio_file)
simplertp.send_rtp_packet(RTP_header, audio, ip, port)
```

ip y *port* son valores que hemos obtenido del cliente (vía *proxy*) mediante la descripción de la sesión (en *SDP*), *audio_file* es uno de los parámetros de configuración del *User Agent* guardados en el XML, y donde *BIT* será un bit aleatorio y *ALEAT* un número entero aleatorio obtenidos a partir de la biblioteca `random` o `secrets` de Python3.

3 Servidor Proxy/Registrar

Debe implementarse un servidor con capacidad de registro de usuarios y que pueda ser utilizado como *proxy*.

El servidor ha de ejecutarse de la siguiente manera:

```
$ python proxy_registrar.py config
```

donde *config* será un fichero XML con la configuración.

Por ejemplo:

```
$ python proxy_registrar.py pr.xml
```

En caso de no ejecutarse correctamente, el programa debería imprimir:

```
Usage: python proxy_registrar.py config
```

En caso contrario, el servidor imprimirá por pantalla algo parecido a la siguiente línea:

```
Server MiServidorBigBang listening at port 5555...
```

El servidor guardará la información de registro en un diccionario⁶.

3.1 Configuración

La configuración por defecto ha de estar almacenada en un fichero XML válido, que deberá seguir las especificaciones indicadas en el DTD dado por los ejercicios alternativos E y F, donde los elementos son los siguientes:

- **server**: Contiene los valores del servidor, incluyendo un nombre (p.ej. *MiServidorGuay*), y la IP y el puerto donde escuchará. En caso de no indicar la IP, se utilizará 127.0.0.1.

⁶Este diccionario ha de ser un atributo de clase no de la instancia, véase <http://www.toptal.com/python/python-class-attributes-an-overly-thorough-guide>

- database: ruta del fichero JSON con la base de datos de usuarios registrados (obligatorio sólo si tienes que hacer el ejercicio H) y ruta del fichero JSON con las claves de los usuarios registrados (obligatorio sólo si tienes que hacer el ejercicio G; el fichero ha de llamarse *passwords.json* y ya se da, vacío, entre los ficheros de inicio de la práctica).
- log: ruta del fichero de *log*.

3.2 Códigos de respuesta

El servidor Proxy/Registrar debería responder con los siguientes códigos de respuesta:

- SIP/2.0 100 Trying: al recibir un INVITE.
- SIP/2.0 180 Ringing: al recibir un INVITE.
- SIP/2.0 200 OK: deberá incluir SDP en el cuerpo (y su correspondiente cabecera).
- SIP/2.0 400 Bad Request: si la petición está mal formada.
- SIP/2.0 404 User Not Found: usuario no registrado
- SIP/2.0 405 Method Not Allowed: si se manda en la petición cualquier otro método diferente a los que sabe manejar.

3.3 Log

El servidor proxy/registrar ha de escribir mensajes de depuración y de *log* en un fichero indicado en el fichero XML de configuración. Cada línea de este fichero de log ha de tener un mensaje con el que se pueda trazar qué es lo que ha ocurrido (o está ocurriendo) en el servidor.

El fichero de log del servidor ha de tener una **estructura idéntica** al que se ha mostrado para el caso del cliente.

3.4 Base de datos de usuarios registrados

El servidor de registro ha guardar los usuarios registrados en cada momento en un fichero JSON. Cada entrada del diccionario contendrá la información relativa a un usuario, en particular, su dirección, su IP, su puerto, la fecha del registro (en segundos desde el 1 de enero de 1970) y el tiempo de expiración (en segundos).

Si no tienes que realizar el ejercicio H, valdrá con escribir en este fichero JSON (será un fichero sólo de escritura, del que nunca se leerá).

4 Ejercicios alternativos

Dependiendo de tu último número de tu DNI / NIE, deberás realizar los siguientes ejercicios que se describen en la Tabla 1.

Últ. nº DNI/NIE	Ej. A	Ej. B	Ej. C	Ej. D	Ej. E	Ej. F	Ej. G	Ej. H
0	X			X		X		X
1	X			X	X		X	
2	X		X			X		X
3	X		X		X		X	
4	X			X		X	X	
5		X	X			X		X
6		X		X	X			X
7		X	X		X		X	
8		X		X		X		X
9		X		X	X		X	

Table 1: Tabla de ejercicios alternativos a realizar según el último número de DNI/NIE del estudiante.

4.1 Ejercicio A

Se debe crear un XML de configuración del UAs a partir del siguiente DTD:

```
<!ELEMENT config (account, uaserver, rtpaudio, regproxy, log, audio)>
<!ELEMENT account EMPTY>
<!ATTLIST account username CDATA #REQUIRED>
<!ATTLIST account passwd CDATA #REQUIRED>
<!ELEMENT uaserver EMPTY>
<!ATTLIST uaserver ip CDATA #IMPLIED>
<!ATTLIST uaserver puerto CDATA #REQUIRED>
<!ELEMENT rtpaudio EMPTY>
<!ATTLIST rtpaudio puerto CDATA #REQUIRED>
<!ELEMENT regproxy EMPTY>
<!ATTLIST regproxy ip CDATA #REQUIRED>
<!ATTLIST regproxy puerto CDATA #REQUIRED>
<!ELEMENT log EMPTY>
<!ATTLIST log path CDATA #REQUIRED>
<!ELEMENT audio EMPTY>
<!ATTLIST audio path CDATA #REQUIRED>
```

4.2 Ejercicio B

Se debe crear un XML de configuración del UAs a partir del siguiente DTD:

```

<!ELEMENT config (account, uaserver, rtpaudio, regproxy, log, audio)>
<!ELEMENT account EMPTY>
<!ATTLIST account username CDATA #REQUIRED>
<!ATTLIST account passwd CDATA #REQUIRED>
<!ELEMENT uaserver EMPTY>
<!ATTLIST uaserver ip CDATA #IMPLIED>
<!ATTLIST uaserver puerto CDATA #REQUIRED>
<!ELEMENT rtpaudio EMPTY>
<!ATTLIST rtpaudio puerto CDATA #REQUIRED>
<!ELEMENT regproxy EMPTY>
<!ATTLIST regproxy ip CDATA #REQUIRED>
<!ATTLIST regproxy puerto CDATA #REQUIRED>
<!ELEMENT log (#PCDATA)>
<!ELEMENT audio (#PCDATA)>

```

4.3 Ejercicio C

Formato de línea de los ficheros log:

Tiempo en formato YYYYMMDDHHMMSS (espacio en blanco) Evento

Nótese que cada línea ha de contener la fecha en el formato indicado, luego un espacio en blanco y luego una de las siguientes. Un ejemplo de un fichero que siga esta estructura es:

```

...
20161128152045 Starting...
20161128153012 Sent to 127.0.0.1:5555: REGISTER sip:leonard@bigbang.org:1234 SIP/2.0 [...]
20161128153017 Received from 127.0.0.1:5555: 200 OK [...]
20161128153036 Sent to 127.0.0.1:5555: INVITE penny@girlnextdoor.com [...]
...
20161128160035 Sent to 127.0.0.1:5555: BYE [...]
20161128160035 Received from 127.0.0.1:5555: 200 OK [...]
20161128160112 Finishing.

```

4.4 Ejercicio D

Formato de línea de los ficheros log:

Tiempo en formato YYYY-MM-DD ss: (espacio en blanco) Evento

donde ss son los segundos desde el inicio del día. Nótese que cada línea ha de contener la fecha en el formato indicado, luego un espacio en blanco y luego una de las siguientes. Un ejemplo de un fichero que siga esta estructura es:

```

...
2016-11-28 55245 Starting...
2016-11-28 55812 Sent to 127.0.0.1:5555: REGISTER sip:leonard@bigbang.org:1234 SIP/2.0 [...]

```

```

2016-11-28 55817 Received from 127.0.0.1:5555: 200 OK [...]
2016-11-28 55836 Sent to 127.0.0.1:5555: INVITE penny@girlnextdoor.com [...]
...
2016-11-28 57635 Sent to 127.0.0.1:5555: BYE [...]
2016-11-28 57635 Received from 127.0.0.1:5555: 200 OK [...]
2016-11-28 67672 Finishing.

```

4.5 Ejercicio E

Se debe crear un XML de configuración del servidor proxy-registrar a partir del siguiente DTD:

```

<!ELEMENT config (server, database, log)>
<!ELEMENT server EMPTY>
<!ATTLIST server name CDATA #IMPLIED>
<!ATTLIST server ip CDATA #IMPLIED>
<!ATTLIST server puerto CDATA #REQUIRED>
<!ELEMENT database EMPTY>
<!ATTLIST database path CDATA #REQUIRED>
<!ATTLIST database passwdpath CDATA #REQUIRED>
<!ELEMENT log EMPTY>
<!ATTLIST log path CDATA #REQUIRED>

```

4.6 Ejercicio F

Se debe crear un XML de configuración del servidor proxy-registrar a partir del siguiente DTD:

```

<!ELEMENT config (server, database, log)>
<!ELEMENT server EMPTY>
<!ATTLIST server name CDATA #IMPLIED>
<!ATTLIST server ip CDATA #IMPLIED>
<!ATTLIST server puerto CDATA #REQUIRED>
<!ELEMENT database EMPTY>
<!ATTLIST database path CDATA #REQUIRED>
<!ATTLIST database passwdpath CDATA #REQUIRED>
<!ELEMENT log (#PCDATA)>

```

4.7 Ejercicio G

Se ha de implementar el mecanismo de registro que se utiliza en SIP (véase práctica 5 o Wikipedia⁷). El mecanismo de registro seguro parte de la asunción que el usuario tiene una cuenta en el servidor registrar. Así, Se detallará un archivo de texto que tendrá en cada línea la dirección de la cuenta y la contraseña (en claro).

⁷http://en.wikipedia.org/wiki/Cryptographic_nonce

En el lado del User Agent, hará uso de la contraseña especificada en el archivo de configuración.

Este mecanismo sólo hace falta implementarlo para el método REGISTER. Todos los UAs deben tener un registro autenticado, aunque en la figura, por simplificar, sólo se hace en uno de los casos.

Asimismo, ha de añadirse la siguiente respuesta al servidor:

- SIP/2.0 401 Unauthorized: cuando un usuario se intenta registrar en el servidor de registro sin autenticarse.

La respuesta del servidor de registro será entonces parecida a la siguiente:

```
SIP/2.0 401 Unauthorized
WWW Authenticate: Digest nonce="898A9898F7989A8989D8989"
```

Cuando se hace la petición en respuesta a un 401 Unauthorized, en las cabeceras del REGISTER ha de incluirse, además de la cabecera con el tiempo de expiración, una cabecera con la información de autenticación.

```
REGISTER sip:leonard@bigbang.org:1234 SIP/2.0
Expires: 3600
Authorization: Digest response="F1231A2321B2312E3212123"
```

Para crear el valor de *nonce* se ha de utilizar funcionalidad de la biblioteca *secrets* de Python3. Para el cálculo de *response*, se puede utilizar la biblioteca *hashlib* de Python3.

4.8 Ejercicio H

a) Se pide añadir cabeceras de Contributing Sources (CSRCs) a los paquetes RTP, como si nuestro envío fuera el resultado de mezclar varias fuentes diferentes. Para ello, se puede hacer uso de la biblioteca *rtpsimple*, pero utilizando el método *setCSRC* para pasarle una lista de CSRCs creados aleatoriamente con la biblioteca de Python3 llamada *random*, tal y como se muestra a continuación:

```
cabeceraRTP = simplertp.RtpHeader()
csrc = [ALEAT1, ALEAT2, ALEAT3, ALEAT4, ...]
cabeceraRTP.set_header(..., cc=ALEATORIO)
cabeceraRTP.setCSRC(csrc)
```

b) Al arrancar, el servidor de registro ha de comprobar si había usuarios registrados en el fichero JSON. Si es así, los incorporará como usuarios registrados a todos los efectos.

5 Capturas

Una vez terminada la práctica, se pide que se realicen las siguientes dos capturas:

5.1 Captura llamada.libpcap

La primera captura es una conversación completa entre dos *User Agents*, incluyendo el envío de audio vía RTP:

1. Registro de dos UA
2. Establecimiento de llamada
3. Envío (bidireccional) RTP de audio
4. Finalización de la llamada.
5. Baja de los UA.
6. La captura original se filtrará para que sólo incluya los paquetes SIP y los cuatro primeros y cuatro últimos paquetes RTP con audio en cada dirección.
7. La captura resultante se guardará en el fichero llamada.libpcap y se subirá al repositorio.

5.2 Captura error.libpcap

La segunda captura consiste en el registro y la baja de un *User Agent*, que hace una llamada a un UA que no está registrado.

1. Registro de UA1
2. Intento de establecimiento de llamada de UA1 (llamando a UA2, que no está registrado)
3. La captura resultante, convenientemente filtrada, se guardará en el fichero error.libpcap y se subirá al repositorio.

6 Preguntas frecuentes

A continuación, se presentan una serie de preguntas frecuentes que los estudiantes suelen tener al realizar la práctica. Por favor, lee esta sección antes de realizar cualquier pregunta en el foro.

- ¿Tengo que crear un UA integrado?

Para la práctica básica no hace falta. Se puede lanzar el UAC y el UAS desde diferentes *shells*. Si alguien implementa un sistema de hilos (*threads*) para tener ambas partes integradas en un único UA, será considerado como un requisito avanzado.

- Cuando estoy enviando vía RTP, tengo el User Agent correspondiente bloqueado y no puedo mandar un BYE. ¿Cómo lo hago?

En la práctica básica, esperaremos a que se termine de enviar la canción. Una vez que se haya terminado el envío, se puede enviar el BYE.

- En los métodos INVITE y BYE tenemos que poner la dirección del receptor, ¿también tenemos que sacarla del XML?

No, la dirección del receptor es un parámetro que le pasamos a *uaclient.py* cuando lo ejecutamos desde la *shell*.

- Me gustaría preguntar si va a ser posible presentar la práctica final en nuestros propios ordenadores.

La práctica ha de funcionar en los ordenadores de los laboratorios docentes, porque es la única manera que tenemos de comprobar que funciona bien. Si además funciona en otros entornos, *ferpecto*.

- ¿Quién tiene que enviar el fichero MP3 por RTP? ¿El que envía la invitación, el invitado, o los dos?

Los dos, cada uno el fichero MP3 al puerto RTP donde la otra parte está esperando para escucharlo.

- En el guión de la práctica, pone que el método REGISTER tiene que mandar el puerto del UAS, pero digo yo ¿Es realmente necesario? ya que esa información la vas a mandar en el SDP del método INVITE, ¿no?

Sí, es estrictamente necesario mandar el puerto del UAS en el REGISTER. Es el puerto donde el proxy se pondrá en contacto contigo cuando alguien haga un INVITE a tu dirección. Por otro lado, el puerto del SDP es el puerto donde el UA quiere recibir el MP3 y este puerto es diferente al del UAS. Recuerda que el del UAS lo programas tú y esperarás que llegue SIP (o sea, texto en claro); no está preparado para recibir MP3 sobre RTP. Por eso, has de especificar un puerto adicional para recibir MP3 que es donde podrás enganchar, si quieres, el vlc para escuchar MP3.

- Cuando me he puesto a capturar el intercambio del fichero, me he encontrado con un problema, al enviar el REGISTER, el puerto que lo envía no es el que debería, sino que es uno al azar, y al comenzar a intercambiar el fichero, el puerto desde el que se produce el intercambio no es el debido si no que es otro al azar y me salta un error diciendo que el puerto no esta conectado o algo parecido.

Cuando abres un socket, el sistema operativo te abre un puerto al azar. Esto no es mayor problema. Sólo los servidores (que están escuchando continuamente) han de estar siempre en un mismo puerto; esto es por una razón lógica: ha de ser un puerto conocido para que alguien se ponga en contacto con ellos.

- Algo que he dado por hecho en mi práctica es que los UA usan conexiones no persistentes para las comunicaciones con el proxy, aunque esto implica crear un socket nuevo para casi cada mensaje entre UA y proxy. De esta forma es un poco lioso seguir las capturas del Wireshark (aparecen con un puerto de envío distinto cada vez) pero no sabía otra forma de que el servidor pudiera escuchar peticiones de registro de varios UA. ¿es una solución válida?

Sí, haces bien al crear siempre un socket nuevo. De lo contrario, tendrías una conexión potencialmente abierta durante mucho tiempo.

- Al capturar la canción veo paquetes UDP con error de checksum. ¿Qué hago mal?

Esto es debido al “check-sum offloading”⁸ y es porque estamos capturando en la misma máquina desde la que emitimos.

Para “solucionarlo” en las preferencias de Wireshark en el protocolo en cuestión debes inhabilitar el cálculo del checksum.

- ¿Cómo utilizo vlc para escuchar lo que me llega?

Tendrías que ejecutar algo parecido a lo siguiente (con su IP y puerto correspondiente, los que pongo aquí son sólo una IP y un puerto de ejemplo), redireccionando la salida de error a la papelera⁹:

```
cvlc rtp://@127.0.0.1:23032 2> /dev/null
```

⁸http://www.wireshark.org/docs/wsug_html_chunked/ChAdvChecksums.html

⁹Según la versión de cvlc puede que haga falta o no la @ en el comando.