SISTEMAS DISTRIBUIDOS Y SERVICIOS WEB

TRABAJO 1 – RMI: SERVICIO RADIUS

DOMINGO FERNÁNDEZ PÍRIZ

FCO. JAVIER ORTIZ BONILLA

▪

 Departamento de Telemática

Índice

1. Descripción del trabajo

1.1. Tema

Realizar un programa cliente servidor para dar soporte a un servicio RADIUS en una red wifi en la que existen varios puntos de acceso que se comunican con un servidor central. Los usuarios se conectan a la red a través de los puntos de acceso, los cuales comprueban las credenciales de los clientes ejecutando métodos en el servidor central para determinar si se pueden conectar y más funcionalidades.

1.2. Servicio RADIUS para autenticación

La finalidad de este proyecto es implementar un servicio RADIUS de modo que un AP al que se conectan clientes pregunte a un servidor RADIUS, mediante invocación de métodos remotos, si el cliente puede entrar o no. Así mismo, el cliente debe implementar métodos que puedan ser llamados de forma remota por el servidor RADIUS para que este pueda realizar operaciones como desconectar a un usuario.

Existen, pues, tres entidades fundamentales en este escenario:

1. Los usuarios.
2. Los puntos de acceso (AP).
3. El servidor RADIUS.

En la realidad un usuario se intenta conectar al AP que tenga más cerca y este le pide sus credenciales (usuario y contraseña). El AP entonces manda estas credenciales al servidor RADIUS, y este, suponiendo que se está usando el protocolo CHAP, envía un desafío de vuelta. El desafío consiste en una cadena aleatoria a la que el AP debe responder haciendo un hash de una nueva cadena que consiste en el nombre de usuario, la contraseña y el desafío concatenados. De esta forma la contraseña no viaja en texto plano por la red ni su hash tampoco. Si la respuesta al desafío es correcta (lo será si la contraseña lo es) el RADIUS responde con un ACCEPT, y en caso contrario, con un REJECT.

Este comportamiento lo podemos ver en la siguiente ilustración.

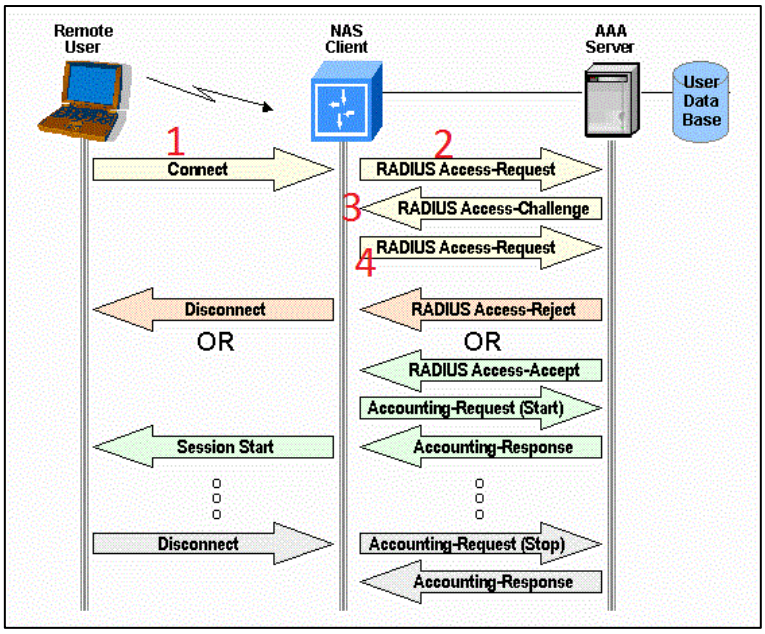


Ilustración 1: Paso de mensajes en una conexión

2. Implementación

Para el desarrollo proyecto, hemos creado en Eclipse un paquete de código que tiene a su vez tres paquetes: cliente, ap y servidor.

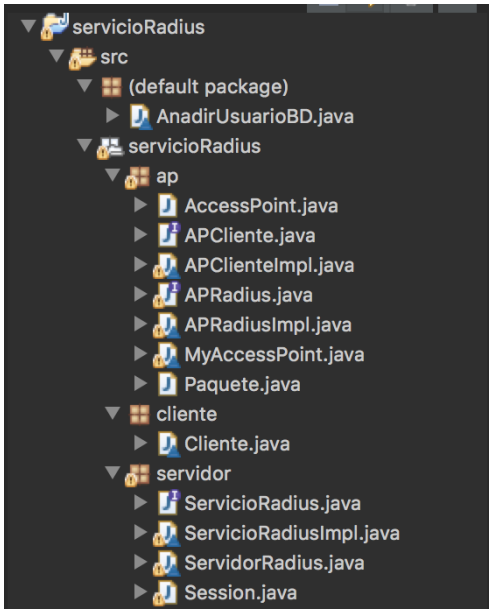


Ilustración 2: Distribución del código

La documentación de las clases puede encontrarse en la carpeta `doc`.

2.1. Punto de acceso

Aunque el servicio RADIUS se limitaría a un diálogo entre los clientes (puntos de acceso) y el servidor, es imprescindible para poder observar el funcionamiento y obtener resultados emular de alguna manera usuarios que quieran conectarse a los puntos de acceso. Para esto vamos a dotar al punto de acceso de otros métodos remotos ajenos al servidor RADIUS que permitirán a un cliente emulado conectarse, desconectarse, enviar paquetes y migrar de un punto de acceso a otro.

Por este motivo, existen las clases `APClienteImpl` y `APRadiusImpl`, y un objeto de clase `AccessPointImpl` contiene en su interior un objeto de clase `APClienteImpl` y otro de clase `APRadiusImpl`. `APClienteImpl` contiene los métodos remotos que serán utilizados por un cliente y `APRadiusImpl` contiene los métodos remotos que serán utilizados por el servidor Radius: de esta forma, el cliente sólo conoce `APCliente` y el servidor RADIUS sólo conoce `APRadius`.

Por último, `MyAccessPoint` es la clase que implementa el main y se encarga de crear un nuevo punto de acceso y hacerlo accesible públicamente a los clientes.

2.2. Cliente

La clase `Cliente` recibe como parámetros una dirección MAC, un usuario y una contraseña. Al ejecutar, se muestra el siguiente menú:

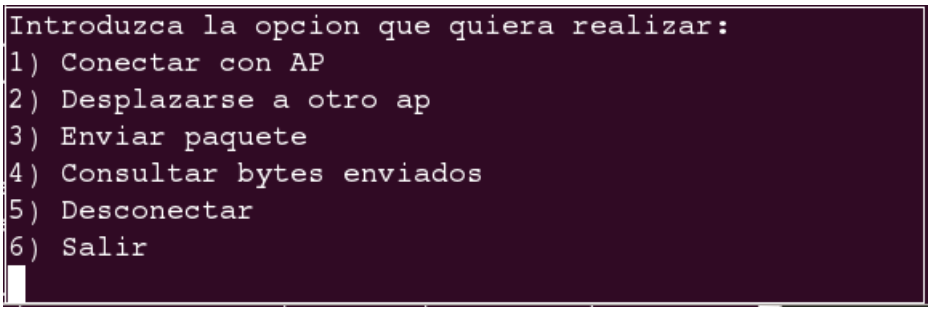


Ilustración 3: Ejecución de Cliente

Cada opción sirve para ejecutar en un AP el método remoto al que se refiere (conectar, desconectar, migrar…).

2.3. Servidor

El servicio RADIUS está implementado en la clase `ServicioRadiusImpl`. Contiene métodos remotos que son llamados por un punto de acceso. Los más importantes son:

- `altaAp` -> Da de alta y guarda la referencia de un nuevo punto de acceso.
- `bajaAp` -> Da de baja y elimina la referencia de un nuevo punto de acceso.
- `accessRequest` -> Un AP solicita la autenticación de un usuario que quiere conectarse. El servidor comprueba que el usuario existe en la base de datos y de ser así retorna un desafío; en caso contrario retorna `null`.
- `challengeResponse` -> Un AP responde a un desafío que había sido retornado por el servidor RADIUS. Si es exitoso y el usuario que quiere conectarse tiene tiempo disponible para navegar, se crea un nuevo objeto `Session` con la información relativa a la nueva sesión del usuario que se conecta y se retorna. En caso contrario, se retorna `null`.
- `moveUser` -> Un AP notifica al servidor RADIUS que un usuario está intentando migrar de otro AP a él. El servidor RADIUS entonces borra en el AP antiguo la sesión de este usuario y actualiza el objeto sesión para este usuario, retornándolo al AP al que el usuario migra.

La clase `ServidorRadius` contiene el main y se encarga de crear un nuevo servicio RADIUS y accerlo públicamente accesible.

2.3.1. Base de datos del servidor RADIUS

`ServicioRadiusImpl` usa una base de datos sqlite llamada **radius.db**. Dentro de esta base de datos existe una tabla usuarios con las siguientes columnas:

- usuario
- contraseña
- expira: Fecha en formato `'yyyy-MM-dd HH:mm:ss'` en la que expira el registro de un cliente. Después de esta fecha no podrá conectarse a la red.
- bytes: Bytes totales transmitidos por el usuario.

```
~$ sqlite3 radius.db

sqlite> CREATE TABLE usuarios (usuario varchar(30)
PRIMARY KEY, password varchar(30) NOT NULL, expira
DATETIME, bytes INT DEFAULT 0);

sqlite> .quit
```

Además, `ServicioRadiusImpl` posee un hilo que cada segundo ejecuta una función que comprueba el tiempo en el que expiran los registros de los usuarios. Si alguno ha expirado, se pone a `null` la columna expira y se desconecta al cliente (si estuviera conectado).

Para poder operar desde java con la base de datos sqlite usamos sqlite-jdbc que podemos obtener de https://bitbucket.org/xerial/sqlite-jdbc/downloads/

Por último, para facilitar la inserción de nuevos usuarios en la base de datos hemos hecho un programa simple en java (`AnadirUsuarioBD`) que acepta como argumentos un nombre de usuario, su contraseña y el tiempo en segundos que dura su registro (desde el momento en que se ejecuta) e inserta esta información en la base de datos.

3. Compilación y ejecución

NOTA IMPORTANTE: Los rmiregistry deben hacerse en el directorio bin del proyecto.

Para la compilación y ejecución se dispone de un makefile con los siguientes targets:

- **compile**: Compila todo el código.

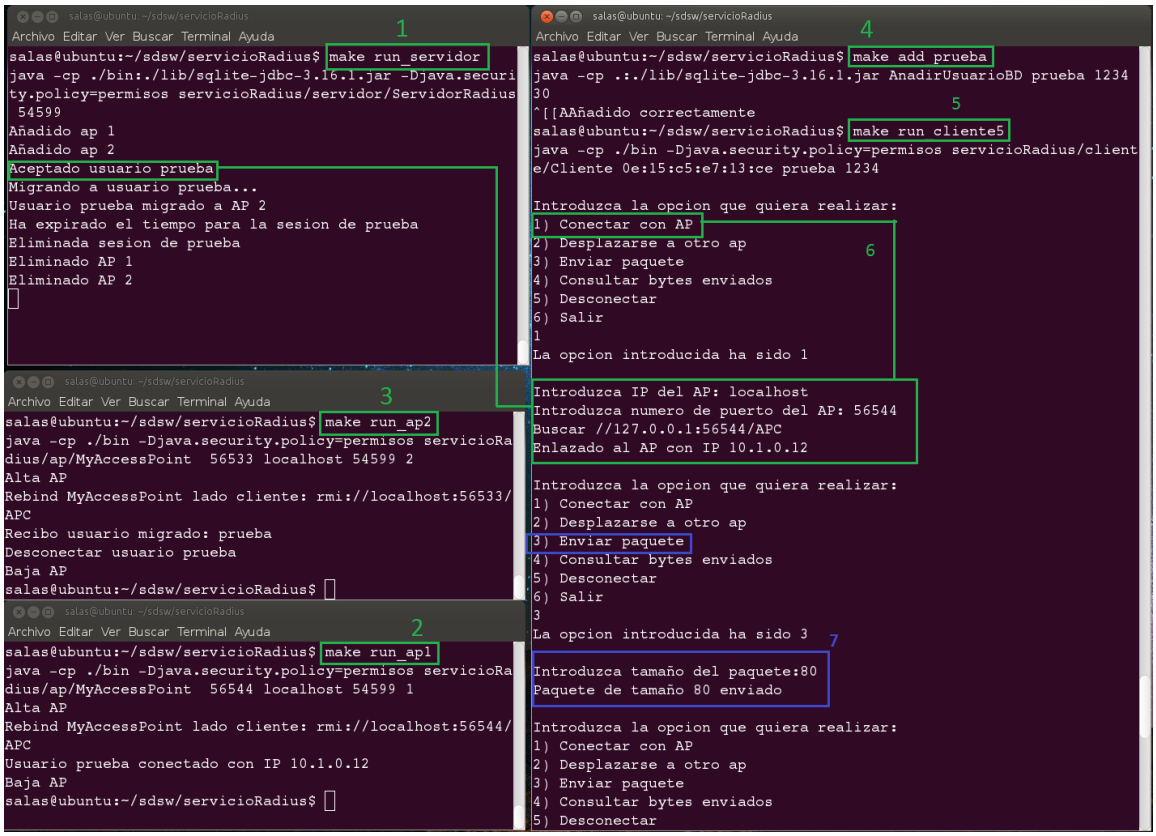
- **run_servidor**: Pone en marcha el servidor con puerto 54599 (se debe, por lo tanto, haber hecho `rmiregistry 54599` previamente).

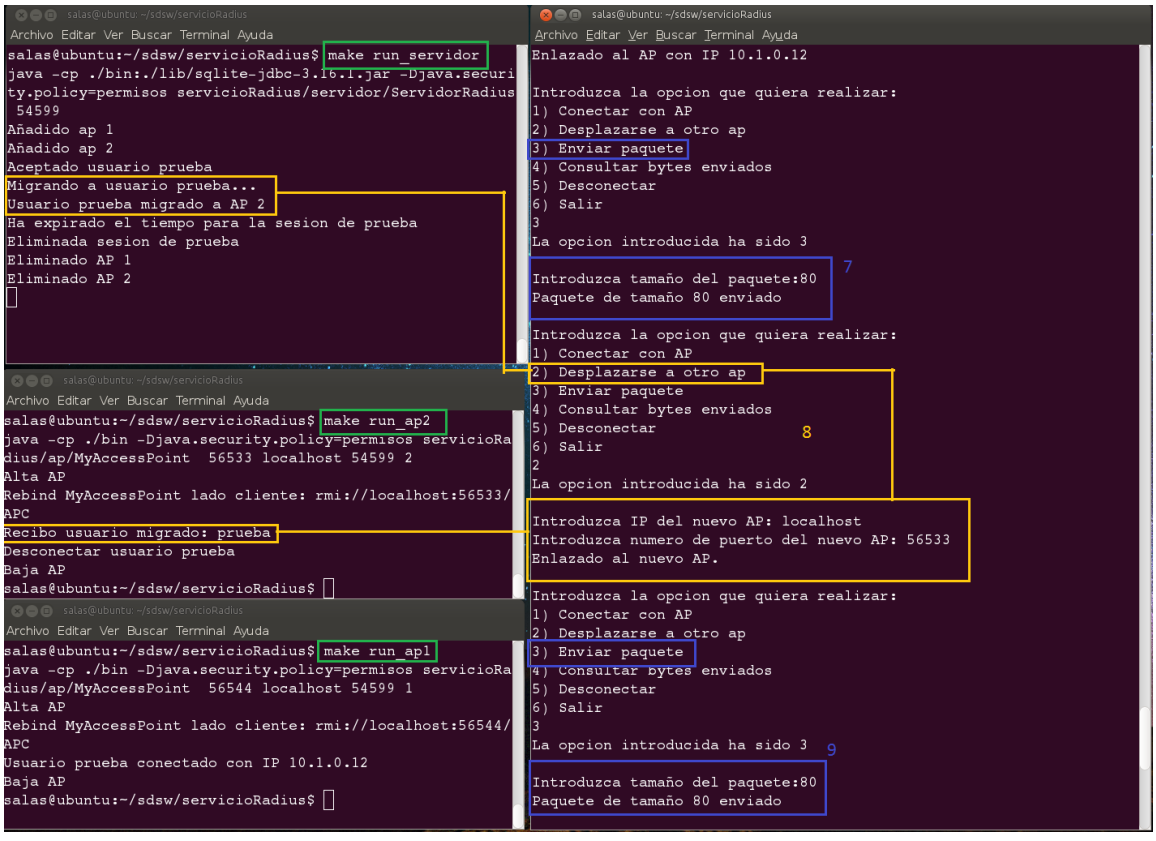
- **run_ap1:** Pone en marcha un AP con id 1 y puerto 56544 (se debe, por lo tanto, haber hecho `rmiregistry 56544` previamente).
- **run_ap2:** Pone en marcha un AP con id 2 y puerto 56533 (se debe, por lo tanto, haber hecho `rmiregistry 56533` previamente).

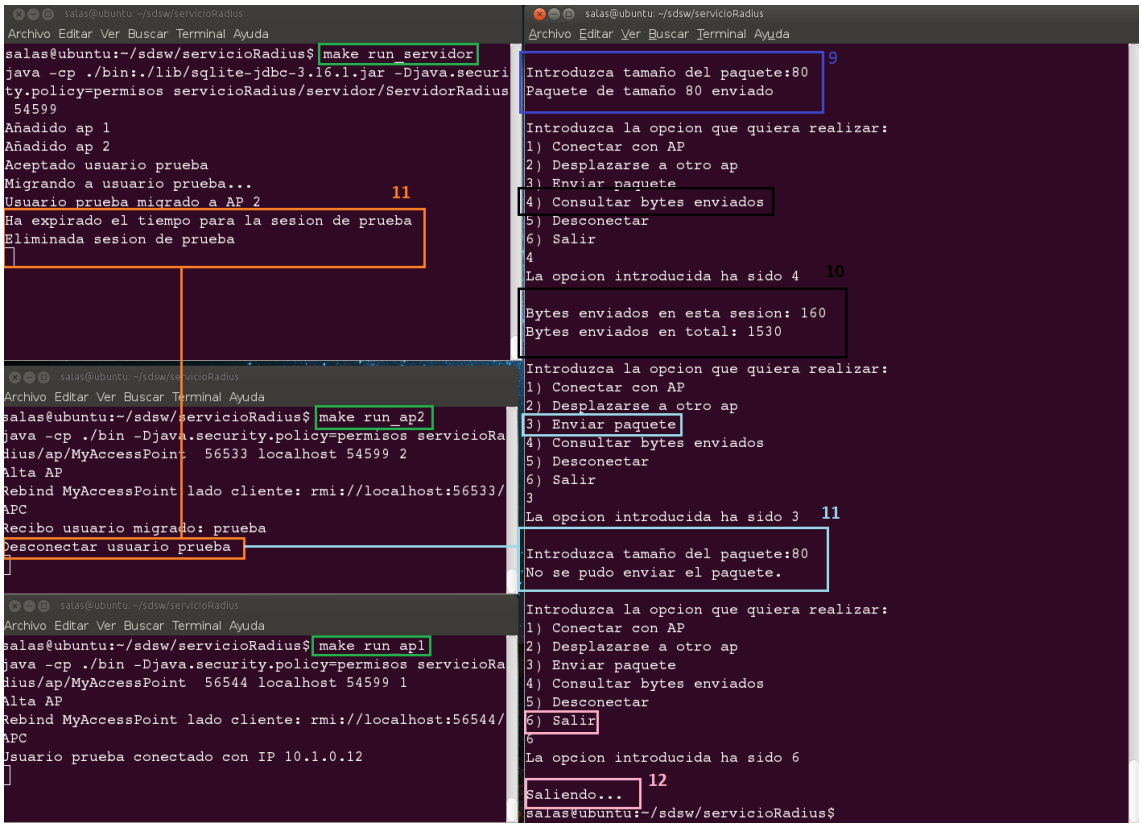
- **run_cliente1:** Pone en marcha un cliente con el usuario 'domin', el cual tiene tiempo de registro hasta 2018.
- **run_cliente2:** Pone en marcha un cliente con el usuario 'javi', el cual tiene tiempo de registro hasta 2018.
- **run_cliente3:** Pone en marcha un cliente con el usuario 'foo', cuyo tiempo de registro ya se agotó, por lo que no puede conectarse.
- **run_cliente4:** Pone en marcha un cliente con un usuario que no existe.

- **add_prueba**: Añade a la base de datos el usuario 'prueba' con contraseña '1234' y con 60s de registro.
- **run_cliente5:** Pone en marcha un cliente con el usuario prueba. Este podrá conectarse y enviar paquetes durante los 60 segundos posteriores a haber hecho `make add_prueba`.

3.1. Ejemplo de ejecución






salas@ubuntu: ~/sdsw/servicioRadius
Archivo Editar Ver Buscar Terminal Ayuda
salas@ubuntu:~/sdsw/servicioRadius$ make run_servidor
java -cp ./bin:./lib/sqlite-jdbc-3.16.1.jar -Djava.securi
ty.policy=permisos servicioRadius/servidor/ServidorRadius
54599
Añadido ap 1
Añadido ap 2
Aceptado usuario prueba
Migrando a usuario prueba...
Usuario prueba migrado a AP 2
11
Ha expirado el tiempo para la sesion de prueba
Eliminada sesion de prueba
salas@ubuntu: ~/sdsw/servicioRadius
Archivo Editar Ver Buscar Terminal Ayuda
salas@ubuntu:~/sdsw/servicioRadius$ make run_ap2
java -cp ./bin -Djava.security.policy=permisos servicioRa
dius/ap/MyAccessPoint 56533 localhost 54599 2
Alta AP
Rebind MyAccessPoint lado cliente: rmi://localhost:56533/
APC
Recibo usuario migrado: prueba
Desconectar usuario prueba
salas@ubuntu: ~/sdsw/servicioRadius
Archivo Editar Ver Buscar Terminal Ayuda
salas@ubuntu:~/sdsw/servicioRadius$ make run_ap1
java -cp ./bin -Djava.security.policy=permisos servicioRa
dius/ap/MyAccessPoint 56544 localhost 54599 1
Alta AP
Rebind MyAccessPoint lado cliente: rmi://localhost:56544/
APC
Usuario prueba conectado con IP 10.1.0.12
salas@ubuntu: ~/sdsw/servicioRadius
Archivo Editar Ver Buscar Terminal Ayuda
9
Introduzca tamaño del paquete:80
Paquete de tamaño 80 enviado
Introduzca la opcion que quiera realizar:
1) Conectar con AP
2) Desplazarse a otro ap
3) Enviar paquete
4) Consultar bytes enviados
5) Desconectar
6) Salir
4
La opcion introducida ha sido 4
10
Bytes enviados en esta sesion: 160
Bytes enviados en total: 1530
Introduzca la opcion que quiera realizar:
1) Conectar con AP
2) Desplazarse a otro ap
3) Enviar paquete
4) Consultar bytes enviados
5) Desconectar
6) Salir
3
La opcion introducida ha sido 3
11
Introduzca tamaño del paquete:80
No se pudo enviar el paquete.
Introduzca la opcion que quiera realizar:
1) Conectar con AP
2) Desplazarse a otro ap
3) Enviar paquete
4) Consultar bytes enviados
5) Desconectar
6) Salir
6
La opcion introducida ha sido 6
12
Saliendo...
salas@ubuntu:~/sdsw/servicioRadius$