Universidad de Valladolid

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

Trabajo de fin de grado

Grado en ingeniería de las tecnologías específicas de telecomunicación
Mención en telemática

Sistema de control de acceso a redes (NAC) basado en SNMP y VLAN

Autor
Darío Adánez Arroyo

Tutor
Federico Simmross Wanttenberg

Mayo 2020

Agradecimientos

En primer lugar quería agradecerle a Federico el haberme permitido realizar este Trabajo con él y haberme ayudado y guiado siempre que me han surgido dudas. Sin su ayuda no habría sido posible conseguir los resultados obtenidos.

Me gustaría agradecer también a todos los compañeros de la carrera, ya que sin su apoyo, unido a los buenos ratos que hemos pasado juntos, el camino hasta llegar aquí habría sido mucho más duro.

A Marta por haber estado animándome en todo momento para que continuara con la elaboración del Trabajo y no me durmiera en los laureles, más aún.

Por último quería agradecer el apoyo y comprensión de mi familia durante toda mi formación académica, ya que si ellos no habría sido posible llegar hasta aquí.

Resumen

El control de acceso a red es un concepto del ámbito la seguridad de redes de computadoras que consiste en la aplicación de los mecanismos y técnicas necesarias para controlar el acceso a la red y prevenir posibles ataques. Hay múltiples soluciones comerciales que tratan este asunto, pero en su mayoría son demasiado generalistas y consumen una cantidad bastante alta de recursos computacionales. Por este motivo, se decide crear una herramienta que separe a los diferentes usuarios conectados a un *switch* en diferentes redes en función de su rol dentro de la red, incluyendo un mecanismo para que el administrador de red pueda consultar el estado del *switch* y quién está conectado en cada momento.

Para lograr este cometido, hay dos tecnologías que resultan fundamentales: SNMP y VLAN. SNMP es un protocolo de gestión de red que, en este caso, va a servir para notificar cuándo se produce un cambio en el estado de uno de los puertos del *switch*. Mediante la creación de VLAN, será posible tener a los usuarios en diferentes redes en función de su rol y aplicar políticas de cortafuegos de forma más fácil.

Abstract

Network access control is an aproach to computer network security that involves the application of the mecanisms and techniques needed to control the access in a network and prevent potential attacks. There are multiple commercial solutions that address this issue, but the most part of them are too general and consume a lot of computational resources. For this reason it is decided to create a tool that separetes the different users connected to a switch in different networks, based on their role whitin the network, that includes a mechanism for the network administrator to check the switch status and who is currently connected.

To achieve this task, there are two technologies that are fundamental: SNMP and VLAN. SNMP is a network management protocol that, in this case, will notify when there is a change in the state of one of the switch ports. By creting VLANs, it will be possible to have users in different networks based on their role and apply firewall policies more easily.

Palabras clave

Gestión de red, SNMP, VLAN, control de acceso, switch.

Índice general

1. Introducción **1**

1.1. Motivación 1

1.2. Objetivos 3

1.3. Fases del trabajo 4

1.4. Estructura de la memoria 5

2. Tecnologías base **7**

2.1. El protocolo SNMP 7

2.1.1. Precedentes y motivación 8

2.1.2. El nacimiento de SNMP 9

2.1.3. Evolución del protocolo 10

2.1.4. Características del protocolo 13

2.2. VLAN 22

2.2.1. Origen 22

2.2.2. Definición y características 23

3. Análisis y diseño **27**

3.1. Planteamiento 27
3.2. Análisis 28
3.2.1. Requisitos del sistema 29
3.2.2. Casos de uso 30
3.3. Diseño 35
3.3.1. Herramientas a utilizar 37
3.3.2. Diseño de la base de datos 38
3.3.3. Diseño de la interfaz gráfica 41
3.3.4. Diseño del controlador 44

4. Desarrollo de la aplicación 47

4.1. Implementación de los casos de uso 47
4.1.1. Implementación de caso de uso LinkUp 47
4.1.2. Implementación del caso de uso LinkDown 49
4.1.3. Implementación de caso de uso CambiaValores 50
4.1.4. Implementación del caso de uso ActualizaBD 53
4.1.5. Implementación del caso de uso MuestraDetallePuertos . . . 55
4.1.6. Implementación del caso de uso AccesoSistema 55
4.1.7. Implementación del caso de uso ActualizaInterfaz 57
4.2. Métodos relevantes 58
4.2.1. Método obtieneMAC 58
4.2.2. Método cambia_vlan 60
4.3. Pruebas 62

4.3.1. Cumplimiento de requisitos 62

4.3.2. Comparativa de los recursos utilizados por la aplicación frente a Packetfence 67

5. Manual para el administrador **69**

5.1. Configuración del *switch* 69

5.2. Servidor DNS y DHCP 70

5.3. Servidor snmptrapd 73

5.4. Servidor web 74

5.5. Interfaces de la máquina 76

6. Conclusiones y líneas futuras **77**

6.1. Conclusiones 77

6.2. Líneas de trabajo futuro 78

Índice de figuras

2.1. Árbol SMI. 16

2.2. Definición del objeto dot1qTpFdbPort. 17

2.3. Ejemplo de intercambio de mensajes una operación set. [Mau05] . . 18

2.4. Ejemplo de intercambio de mensajes una operación get [Mau05]. . . 18

2.5. Ejemplo de intercambio de mensajes de una operación getbulk [Mau05]. 20

2.6. Ejemplo de envío de trampa [Mau05]. 21

2.7. Agrupamiento lógico con VLAN 24

3.1. Diagrama de casos de uso. 36

3.2. Boceto de la interfaz principal. 43

3.3. Boceto de la ventana para modificar la base de datos. 44

3.4. Diagrama de clases simplificado de la aplicación. 46

4.1. Método ObtieneMAC 48

4.2. Envío de señal UNIX SIGUSR1. 49

4.3. Cambio en la base de datos por desconexión. 50

4.4. Obtención y cambio de VLAN. 50

4.5. Creación del cuadro de dialogo de la clase Gestion. 51

4.6. Creación de etiqueta y QLineEdit para la dirección IP. 52

4.7. Obtención y muestra de los datos actuales de configuración. 52

4.8. Ventana de configuración. 53

4.9. Conexión de señal con su *slot*. 53

4.10. Actualización del campo *mac*. 54

4.11. Obtención y relleno de la tabla que muestra los puertos. 56

4.12. Resultado final de la tabla que muestra el estado del *switch*. 56

4.13. Código para actualizar un puerto de la interfaz. 57

4.14. Conexión de señal UNIX con su *slot*. 58

4.15. Desarrollo del método ObtieneMAC. 59

4.16. Método cambia_vlan. 61

4.17. Aspecto de la ventana principal de la aplicación. 63

5.1. Fichero de creación del servicio dnsmasq. 71

5.2. Configuración de dnsmasq para el resto de redes. 72

5.3. Configuración de dnsmasq para la VLAN restringida. 72

5.4. Fichero de configuración snmptt.conf. 74

5.5. Obtención de dirección MAC y puerto. 75

5.6. Ejemplo de definición de interfaz en el fichero /etc/network/interfaces. 76

Índice de tablas

2.1. Tipos de datos definidos por la SMI [Mau05, McC88b] 15
2.2. Códigos de error SNMP 19
2.3. Tipos de trampa SNMP [Cas90]. 22

3.1. Requisitos funcionales. 30
3.2. Requisitos no funcionales. 30
3.3. Caso de uso LinkUp. 31
3.4. Caso de uso LinkDown. 32
3.5. Caso de uso CambiaValores. 32
3.6. Caso de uso InicializaBD. 33
3.7. Caso de uso MuestraDetallePuertos. 34
3.8. Caso de uso AccesoSistema. 34
3.9. Caso de uso ActualizaInterfaz. 35

Capítulo 1

Introducción

Este capítulo sirve de introducción al trabajo y expone la motivación del mismo, además de enumerar los objetivos que pretende cumplir, comentar la metodología que ha sido empleada y realizar una breve descripción de la estructura de la memoria.

1.1. Motivación

Hoy en día la posibilidad de estar conectado a la red se ha convertido en algo esencial para la gente. Ya sea por motivos de trabajo o para pasar el tiempo libre, la gente pasa varias horas al día conectada a Internet. Según un estudio realizado por Google, el 80 % de las personas que hacen uso de Internet a diario utilizan su teléfono móvil, frente al 67 % que lo hace mediante un ordenador [Goo16]. A pesar de no ser el principal medio utilizado por los consumidores para acceder a Internet, los ordenadores siguen gozando de una cuota de uso bastante alta, como refleja el estudio.

Debido a esta necesidad, es fácil encontrar lugares que proporcionen acceso a internet, ya sea mediante accesos inalámbricos, o mediante una conexión cableada, como por ejemplo el laboratorio 10 de la ETSIT, donde los alumnos pueden conectarse a la red con su ordenador. Estos son lugares donde un usuario llega con su ordenador y lo conecta, o utiliza uno de los que haya en la sala, si es que los hay, y realiza las tareas que desee. Estos puntos de conexión cableada a Internet son fáciles de encontrar en universidades o centros escolares, con el objetivo de proveer

a los alumnos de un espacio donde poder trabajar en las diferentes tareas planteadas en clase, aunque también son usuales en lugares públicos como aeropuertos o estaciones.

En muchos de estos lugares el acceso a la red no es libre, sino que el acceso está limitado a un determinado grupo de usuarios que cumplan ciertos requisitos, como por ejemplo, pertenecer a la facultad donde se encuentra la red de acceso o simplemente estar registrado.
Para poder acceder, el usuario debe autenticarse con unas credenciales que previamente le hayan sido otorgadas. Es por esto que se hace imprescindible que el administrador de la red implante un control de acceso para permitir la conexión únicamente a ese grupo de usuarios autorizados. Además, sería deseable también conocer en tiempo real qué máquinas están conectadas a la red y diferenciar a los clientes para tratar el tráfico con un cortafuegos de manera más fácil. Para solucionar este problema es necesario establecer un mecanismo de control de acceso a red.

El control de acceso a red o NAC *(Network Access Control)* por sus siglas en inglés, es un concepto enmarcado en el ámbito de la seguridad de red, que consiste en la aplicación de las técnicas y mecanismos necesarios para controlar el acceso a una red y prevenir posibles ataques, asegurando de esta forma que todos los dispositivos conectados a una determinada red cumplen una serie de requisitos previamente establecidos por el administrador.

En el mercado existen diversas soluciones a este problema como Nagios y Packetfence [Inv20, Nag20]. Se trata de soluciones muy completas pero también demasiado generalistas, puesto que además de las necesidades a cubrir expuestas anteriormente, incluyen dentro de la distribución otras funcionalidades que nada tienen que ver con el objetivo perseguido. Además, si nos fijamos en los requisitos necesarios para instalar la última versión de Packetfence, por ejemplo, nos encontramos con que es necesario disponer de un servidor con al menos 8 GB de memoria RAM y 100 GB de espacio en disco, sumado a la necesidad de tener como sistema operativo Debian 9.0, Community ENTerprise Operating System *(CentOS)* 7.x o Red Hat Enterprise Linux 7.x Server. De lo contrario será necesario instalar ZEN *(Zero Effort NAC)*, una distribución de Packetfence preconfigurada para ser ejecutada en una máquina virtual [Inv19].

Debido a las características de estos productos, unidas a los altos requisitos computacionales que exigen, puede resultar útil diseñar un sistema de control de acceso que cumpla las necesidades del administrador y que además pueda ser ejecutado en una máquina con pocos recursos.

Este trabajo dependerá de muchas tecnologías y protocolos, pero dos destacan por encima del resto debido a su importancia en el desarrollo de la herramienta. En primer lugar, se utilizará el protocolo SNMP *(Simple Network Manager Protocol)*. Es un protocolo de gestión de red que permitirá conocer los cambios que se produzcan en el *switch* debido a la conexión o desconexión de un dispositivo conectado. En segundo lugar, la presencia de VLAN *(Virtual Local Area Network)* permitirá la separación lógica en diferentes redes de los usuarios conectados, ya que este protocolo permite crear diferentes redes en un mismo dominio de colisión [Cas88, Ins]. En virtud de lo expuesto, este trabajo pretende dar una solución a las necesidades antes expuestas, sin llegar a ofrecer el abanico de posibilidades y potencia de las principales soluciones comerciales existentes, implementando un mecanismo de control de acceso a red *(NAC)* basado en SNMP y VLAN, que diferencie a los usuarios conectados a la red en función de si son conocidos o no, además de proporcionar una herramienta gráfica que permita ver el estado de los puertos del *switch* y quién está conectado.

1.2. Objetivos

El objetivo principal del presente Trabajo de Fin de Grado es el desarrollo de una herramienta que implemente control de acceso a red basado en SNMP y en VLAN, que permita separar a los usuarios en varias redes de acuerdo a sus credenciales, y que proporcione una herramienta gráfica para facilitar su uso. Por lo tanto, es necesario conseguir una herramienta que sea capaz de advertir los cambios ocurridos en el *switch* que proporcionará acceso a los usuarios, mediante la utilización del sistema de trampas del protocolo SNMP y realizar la separación lógica de los usuarios conectados al *switch* mediante la creación de VLAN. Para lograr este objetivo, es necesario conseguir una serie de objetivos menores que conforman el objetivo principal:

- **Detectar el cambio en un puerto**: es necesario detectar la conexión o desconexión de un puerto y asignar a la máquina conectada a ese puerto la VLAN correspondiente.
- **Permitir la autenticación de un usuario:** el sistema debe ser capaz de proporcionar un método de autenticación de usuarios y llevar a cabo los procedimientos necesarios en el caso de que un usuario se autentique correctamente.
- **Proporcionar la configuración de red a los equipos necesarios:** es

necesario proporcionar a las máquinas conectadas parámetros de red tales como una dirección IP, nombre de dominio, servidor DNS, puerta de enlace, etc.

- **Mostrar cambios en tiempo real:** la herramienta gráfica debe de ser capaz de cambiar su estado cuando suceda un cambio en el *switch*.

- **Configuración de los parámetros del *switch*:** la configuración de la red en un momento dado puede cambiar, por lo que programa debe proporcionar un método que permita cambiar los parámetros del *switch* fundamentales para la aplicación, tales como la dirección IP del dispositivo, el número de puertos o el identificador de VLAN de las redes configuradas en el sistema.

1.3. Fases del trabajo

Para cumplir los objetivos expuestos en el anterior apartado se han completado una serie de fases o etapas que se exponen a continuación:

- **Análisis de las soluciones existentes:** en primer lugar se ha hecho un estudio de las soluciones existentes que intentar dar una solución al problema que motiva este Trabajo de fin de grado.

- **Análisis de objetivos:** En segundo lugar se han definido los objetivos que se pretenden alcanzar y que han sido detallados en la anterior sección.

- **Análisis y estudio del protocolo SNMP y VLAN:** debido a que son los pilares fundamentales sobre los que se sustenta este TFG, ha sido necesario realizar un estudio de ambas tecnologías para profundizar en los conocimientos adquiridos a lo largo de la carrera, poniendo especial hincapié en el funcionamiento de las trampas SNMP.

- **Elección de las tecnologías auxiliares a emplear:** para la elaboración de la herramienta es necesario emplear diversas tecnologías además de SNMP y VLAN, que serán mencionadas más adelante, que representan la base del trabajo. Por lo tanto, es necesario realizar un análisis con la pretensión de conocer y elegir las diversas opciones de las que se dispone.

- **Diseño:** se ha realizado un estudio de los requisitos a cumplir por la aplicación y se ha realizado el diseño de las diferentes partes que lo van a componer.

- **Desarrollo de la aplicación:** fase en la que se desarrolla la aplicación, tanto la parte programática como la configuración y puesta en marcha de los servidores necesarios para hacer funcionar la herramienta.
- **Pruebas:** fase de pruebas para comprobar el cumplimiento de los objetivos marcados y la ausencia de errores. En esta fase, además, se han corregido algunos errores de diseño, al igual que algunos aspectos han sido mejorados.

1.4. Estructura de la memoria

En este apartado de la introducción se intentará explicar y detallar en contenido de la memoria. La memoria consta de un total de 6 capítulos, que se detallan a continuación, a excepción del presente capítulo, que está próximo a finalizar. Además, algunos capítulos abordan una temática relacionada entre sí, por lo que es posible hablar de 4 bloques temáticos distintos en la memoria.

- **Segundo capítulo:** este capítulo realiza una introducción al protocolo SNMP y al concepto de VLAN, explicando el motivo de el surgimiento de estas tecnologías y haciendo un repaso histórico a su evolución.
- **Tercer capítulo:** muestra el proceso en la fase de diseño de la aplicación llevado a cabo para conseguir los objetivos descritos en el presente capítulo.
- **Cuarto capítulo:** aborda la fase de implementación de la aplicación y las pruebas realizadas.
- **Quinto capítulo:** en este capítulo se detallan los pasos que el administrador debe dar para poner en funcionamiento la herramienta.
- **Sexto capítulo:** el último capitulo detalla las conclusiones extraídas tras la realización del Trabajo de Fin de Grado, así como se extraen diversas líneas de futuro con las que ampliar y mejorar la aplicación.

Como se puede apreciar, los capítulos 1 y 2 están relacionados entre sí, ya que sirven de introducción. En el caso del primer capítulo, realiza la introducción al trabajo en su conjunto, mientras que el capítulo 2 realiza una introducción a la las tecnologías a usar en el desarrollo del TFG.

El tercer capítulo constituye un bloque temático en sí mismo, ya que realiza el análisis de los requisitos de la aplicación y aborda la fase de diseño.

El tercer bloque temático está formado por los capítulos 4 y 5, puesto que abordan la fase de desarrollo de la herramienta, desde la implementación del diseño realizado en el bloque anterior, hasta la puesta en marcha de los servidores necesarios para el funcionamiento de la red.

El sexto y último capitulo constituye por si mismo un bloque temático, ya que menciona las conclusiones sacadas de la elaboración del trabajo y las líneas de desarrollo futuras que pueda tener la herramienta.

Capítulo 2

Tecnologías base

En este capítulo se realizará una introducción teórica a las tecnologías base a utilizar en el presente Trabajo de Fin de Grado, que resultan fundamentales para la consecución de los objetivos marcados en el anterior capítulo. Se presentará un marco teórico para poner en contexto al lector acerca de dichas herramientas. De este modo, en primer lugar se pondrá énfasis en las causas de la aparición de ambos protocolos, para continuar exponiendo su evolución y estado actual, además de explicar sus características.

Al finalizar este capítulo se pretende que el lector se encuentre familiarizado con el protocolo SNMP y con el concepto de VLAN, así como con sus notaciones y nomenclatura, para poder comprender mejor el motivo de la elección de estas herramientas para la realización del trabajo.

2.1. El protocolo SNMP

En esta sección del capítulo 2 se va a realizar una introducción teórica del protocolo SNMP, comenzando por explicar el motivo de su aparición, pasando por su evolución histórica hasta acabar hablando sobre su estado y uso actual, además de enumerar sus características y explicar su funcionamiento.

2.1.1. Precedentes y motivación

Allá por los años 70, en el inicio de la interconexión de redes, el concepto de monitorización de red era prácticamente desconocido. Cuando ocurría un error en la red, la forma que tenía un administrador de red que había ocurrido era acudir a la máquina para ver había ocurrido. Si bien es cierto que se disponía de algunas herramientas como por ejemplo ICMP *(Internet Control Message Protocol)*, estas eran a veces insuficientes debido a sus limitaciones y al crecimiento que comenzaban a experimentar las redes a finales de los años 80 [Ily19]. En [Ros81], tras la primera gran caída de ARPANET *(Advanced Research Projects Agency NETwork)* en el año 1980, se recoge una discusión acerca de la necesidad de administrar de una forma más sistemática las redes de comunicaciones, ante la complejidad de hacerlo de forma manual, debido al gran aumento experimentado.

Ante la necesidad de implementar mecanismos de monitorización de las redes, comenzaron a surgir diversas soluciones propietarias, con el inconveniente de que solo funcionaban en sus propias redes, impidiendo así la interoperabilidad entre redes que utilizaban diferentes topologías o tecnologías [Wil88].

En noviembre de 1987 nace el protocolo SGMP *(Simple Gateway Monitoring Protocol)* para intentar paliar los problemas originados por la dificultad de realizar la gestión de redes y proponer un protocolo único que evite la heterogeneidad provocada por las soluciones propietarias. En [Dav87] se definen tres objetivos de diseño del protocolo, para lo que se aplica un total de ocho estrategias de diseño para tratar de ajustarse a las necesidades que debía satisfacer el protocolo. Estos objetivos son:

- **Primer objetivo:** minimizar el número de *gateways* o puertas de enlace así como el número de funciones que realizan.
- **Segundo objetivo:** conseguir una utilidad de control y monitoreo lo suficientemente extensible como para cubrir posibles futuras funciones de las puertas de enlace.
- **Tercer objetivo:** conseguir un diseño lo más independiente posible de las arquitecturas de red y los elementos que conforman la red.

En definitiva, esta propuesta nace con el objetivo de aportar una solución a las necesidades de incorporar mecanismos de gestión de red de una forma simple, escalable e idependiente de la arquitectura y *hardware* utilizados, aunque también es una solución temporal, puesto que sus autores la definen como «una respuesta

provisional a la inmediata necesidad de monitorizar las puertas de enlace, mientras se trabaja en una solución más robusta y elaborada» [Dav87].

2.1.2. El nacimiento de SNMP

Debido a la temporalidad anunciada por los autores de SGMP, se desarrolla SNMP como una mejora de SGMP, siendo desarrolado por los mismos autores [Cas88]. Aunque SNMP no es la única propuesta que apareció, ya que surgieron otras alternativas como HEMS *(High-Level Entity Management System)* o CMIP *(Common Management Information Protocol)* [Par87, Ros90]. Este último, a diferencia de SNMP, está basado en la pila de protocolos OSI *(Open System Information)*.

La IAB *(Internet Architecture Board o IAB)* establece que SNMP debe ser adoptado como una solución a corto plazo, a la espera de que se realice la migración a redes OSI, aunque finalmente esta no se produjo y SNMP se convirtió en la solución más utilizada, en parte por su rápido desarrollo frente a HEMS. Así mismo, establece la necesidad de definir una base común de información, así como la idoneidad de que CMIP/CMIS sea compatible con redes TCP/IP, para así poder ser utilizado incluso antes de que la transición entre modelos fuera una realidad. Para lograr sus pretensiones, la IAB instó a los diferentes grupos de trabajo a coordinarse para lograr los siguientes objetivos [Cer88]:

- Convergencia y desarrollo rápido de SNMP.
- Rápida convergencia y definición de la MIB para TCP/IP.
- Disposición para en un futuro realizar la transición de SNMP a CMIP/CMIS.
- Rápida demostración de la capacidad de CMIP/CMIS para utilizar la base definida para TCP/IP.

Como se comentó anteriormente, se pretendía que tanto SNMP como CMIP utilizaran el mismo sistema de base de datos de los elementos gestionados, como estrategia para la convergencia de ambos protocolos. Debido a esta circunstancia, se crean las definiciones de SMI *(Structure of Management Information)* y de lo que hoy se conoce como MIB *(Management Information Base)*. SMI define la estructura y la información contenida en las bases de datos además de los objetos que las forman, definiendo los diferentes parámetros y tipos que contienen [McC88b], mientras que las MIB establecen el esquema de los objetos de la base de datos [McC88a].

Si bien el en un inicio se pretendía que tuviera un corto recorrido, debido a la intención de migrar a redes OSI, es un protocolo muy utilizado a día de hoy, y que ha ido evolucionando y presentando diferentes versiones con el objetivo de pulir sus defectos.

2.1.3. Evolución del protocolo

SNMP ha experimentado una progresiva evolución desde su aparición en 1988. Su puesta en marcha fue muy rápida, ya que no estaba condicionado a la compatibilidad con OSI, como sí lo estaba CMIP/CMIS. Muy pronto fue implementado por la mayoría del mercado, así que se hizo necesaria la revisión de sus características para así poder solucionar los aspectos negativos de su diseño.

SNMPv1

Esta es la primera versión del protocolo, que fue definida en [Cas88], y que como se ha comentado anteriormente se trata de una evolución del protocolo SGMP. Así pues, en esta primera versión existen las operaciones `get`, `set` y `trap` de SGMP definidas en [Dav87], además de la operación `getnext`.

Sin lugar a dudas, la novedad principal en esta primera versión del protocolo con respecto a su antecesor es la aceptación de los conceptos MIB y SMI, cambiando por completo la estructura y acceso a la información. Adopta también las trampas definidas para SGMP, aunque redefine la trampa *Link Failure*, como *Link Down* e implementa dos nuevas trampas: *Link Up Trap*, cuyo cometido es avisar de la activación de un puerto, y *enterpriseSpecific Trap*, con el objetivo de permitir a los fabricantes crear sus propias notificaciones. Además reserva el puerto 161 UDP para que el agente SNMP reciba las solicitudes y el puerto 162 UDP para la recepción de notificaciones desde dichos agentes.

La seguridad del protocolo se basa en las llamadas *communities*, que no son más que contraseñas que se transmiten en texto plano. Un agente SNMP está configurado con tres comunidades básicas: lectura, lectura-escritura y trampas [Mau05].

Un año después de su aparición, la definición de la primera versión quedó obsoleta por la publicación del RFC 1098. Esta actualización se debe a la recomendación hecha por la IAB, que establece la necesidad de que SNMP y CMIP/CMIS, denominado CMOT *(CMIT Over TCP/IP)* en su versión compatible con TCP/IP,

compartan la base de datos de información de gestión [Cas89]. La última modificación realizada sobre esta versión del protocolo se recoge en el RFC 1157 , que recoge «algunas correcciones tipográficas menores» [Cas90].

SNMPv2

La segunda versión del protocolo fue lanzada en 1993, y aún sigue siendo muy utilizada a día de hoy. Está definida en el conjunto de RFCs que van del 1441 al 1452. Se trata de una revisión de la primera versión del protocolo, incluyendo diversas mejoras en rendimiento, seguridad y confidencialidad.

Con respecto a las operaciones que soporta el protocolo, cuenta con las de las versión anterior, además de incorporar una nueva operación, `getbulk`, que se presenta como una alternativa para recuperar grandes cantidades de datos de gestión en una sola solicitud, en lugar de tener que concatenar varias operaciones `getnext` [Cas93a] y reformula el concepto de trampa para pasar a llamarlo notificación.

Sin ambargo, el principal cambio fue el que afectaba a seguridad en las comunicaciones y a la autenticación de usuarios, que en la anterior versión estaba basada en *communities* y ahora pasaba a sostenerse sobre el cálculo de un resumen del mensaje que se añadía a la PDU. Este resumen estaba generado a partir de una clave secreta conocida únicamente por el emisor del mensaje y el agente SNMP [Gal93]. Este cambio hacía compleja la autenticación de los usuarios, y fue rechazado por gran parte de la comunidad, que prefería la autenticación basada en comunidades debido a su sencillez. Es por ello que SNMPv2 fue reformulado en el año 1996, y fue dividido en dos subversiones: SNMPv2c y SNMPv2u. La primera de ellas vuelve a utilizar las comunidades de SNMPv1 como mecanismo de autenticación y fue rápidamente adoptada por el mercado, mientras que en el segundo caso la autenticación está basada en usuarios y apenas tuvo éxito.

Todos estos cambios introducidos en la nueva versión del protocolo vienen de la mano del desarrollo de la segunda versión de SMI y MIB, que introduce nuevas ramas al árbol MIB, como por ejemplo la rama *snmpV2* y nuevos tipos de datos a los ya existentes previamente. Además, la definición de un objeto en SMI II cambió, añadiendo la posibilidad de introducir nuevos campos que permitan un mayor control en el acceso y en la adición de objetos [McC91, Cas93b].

La mayor aceptación de SNMPv2c, hace que en aspectos de seguridad el protocolo quede prácticamente igual que en la versión inicial, por lo que estos aspectos

tenían que continuar siendo suplidos por otros mecanismos ajenos al diseño del protocolo, motivo fundamental para que se planteara la creación de una nueva versión del protocolo.

SNMPv3

La seguridad en SNMP había sido el principal quebradero de cabeza desde su aparición. Con la aparición de SNMPv2 se intentó mitigar este defecto, pero la vuelta a la autenticación basada en comunidades de SNMPv2c, dejaba un protocolo inseguro, puesto que un sistema de seguridad basado en contraseñas que se mandan en texto plano deja bastante que desear. Por este motivo se decide implementar SNMPv3, una versión del protocolo que sale a la luz en 2002, definido en el conjunto de RFCs que se van del 3411 al 3418 [Mau05].

El principal cambio que introduce esta versión es el abandono de la notación de *manager* y agente SNMP, para hablar de *entidades* SNMP. Una entidad está compuesta por un *motor* SNMP y una o varias aplicaciones. Este nuevo concepto, unido al cambio en la terminología, hicieron que, a pesar de realizar las mismas operaciones que su predecesor, pareciera un protocolo distinto, en vez de una evolución, produciendo el rechazo de la comunidad [Har02].

En cuando a la seguridad, principal motivación de esta versión, el protocolo cuenta con numerosos cambios. El primero de ellos es que ahora el intercambio de claves se realiza utilizado algoritmo Diffie-Helman, quedando así resuelto el problema de la contraseña transmitida en texto plano [Joh00]. Se definen también dos nuevos mecanismos de seguridad:

- **USM *(User-based Security Model).*** Tiene su origen en la definición de SNMPv2u. Proporciona autenticidad en el mensaje utilizando un HMAC *(Hash Message Authentication Code)* combinado con una función *hash* MD5 o SHA-1. Además, proporciona confidencialidad a los mensajes SNMP utilizando un cifrado AES y un control temporal con el que protege al mensaje contra retrasos [Blu02].

- **VACM *(View Access Control Model).*** Es un sistema que define el acceso a una rama del árbol MIB, en función del cumplimiento o no de una serie de reglas y condiciones, en función del lo que se esté pidiendo y quien lo esté pidiendo [Wij02].

2.1.4. Características del protocolo

Una vez realizado un breve repaso por la historia del protocolo, es necesario entrar en materia y explicar su funcionamiento, así como los elementos que lo componen y las operaciones que realiza.

SNMP es una protocolo de la pila de protocolos TCP/IP definido por la IETF *(Internet Engineering Task Force)* que opera en la capa de aplicación. Su función es facilitar el intercambio de información entre dispositivos de red, como *routers*, *switches* o simples *hosts*, para así poder gestionar y supervisar los elementos que componen una red. A pesar de contar con tres versiones diferentes, todas ellas son compatibles con sus predecesoras.

En una red administrada con SNMP, se puede hablar de la existencia de tres componentes [Mau05]:

- **Dispositivos de red:** son los dispositivos que conforman la red y se encargan de recoger y almacenar la información. Contienen un agente SNMP. Estos dispositivos pueden ser *switches*, *routers*, *hubs*, ordenadores, etc.
- **Agentes SNMP:** son módulos *software* contenidos en los dispositivos de red. Conocen la información relevante del dispositivo en el que están contenidos y organizan la información de forma jerárquica.
- **Servidor SNMP:** también se conoce como NMS *(Network Managment System)*. Es el encargado de supervisar y controlar los dispositivos de la red, además de proporcionar los recursos necesarios para ello. En una red gestionada por SNMP puede haber más de un NMS.

En cuanto a las comunicaciones entre los diferentes componentes de la red, se utiliza UDP, por lo que SNMP no es un protocolo orientado a conexión. Esto tiene sentido ya que el uso de TCP añadiría una sobrecarga en la red innecesaria. Los agentes SNMP escuchan en el puerto 161, mientras que se reserva el puerto 162 para que el NMS escuche las diferentes notificaciones que le puedan llegar.

SMI, MIB y OID

Estos tres conceptos definen la forma de manejar la información por parte de SNMP. En primer lugar, se va a explicar qué es **SMI**, puesto que en párrafos

anteriores se ha mencionado su existencia e importancia, pero no se ha llegado a concretar en que consiste y por qué es tan importante en el desarrollo de SNMP.

La estructura de gestión de la información o SMI, porporciona una forma de definir los objetos gestionados y determinar cual va a ser su comportamiento. Un agente SNMP tiene una lista con todos estos objetos y trata de obtener su estado, conformando entre todos el estado del dispositivo gestionado. Define la estructura que deben de tener estos objetos, y los tipos de datos que pueden contener. Para definir todos estos datos, utiliza la notación **ASN.1** *(Abstract Syntax Notation One)*, que se trata de una forma de definir cómo se deben convertir y enviar los datos, permitiendo así la comunicación entre diferentes dispositivos con sistemas operativos distintos sin preocuparse de tener que realizar la conversión de tipos [Mau05].

Como ya se ha comentado, SMI define distintos tipos de datos, que son empleados por MIB y podemos ver en la tabla 2.1.

La base de información de gestión o **MIB**, es una base de datos que contiene los objetos gestionados que un agente SNMP se encarga de obtener. Cualquier parámetro al que un NMS pueda acceder está definido en una MIB. Estos objetos están definidos en forma de árbol jerárquico, como se puede ver en la figura 2.1. En la figura 2.2 se puede observar la definición del objeto *dot1qTpFdbPort*, que será útil para el desarrollo de la aplicación.

Por último, para poder acceder a un objeto de árbol se utiliza su **OID** *(Object IDentifier)*, que no es más que un identificador que relaciona el objeto de forma unívoca. Generalmente se expresa de forma numérica, aunque también se puede expresar con el nombre del objeto. Está formado por los identificadores de cada nodo del árbol, ya sea numérico o en texto, por los que hay que pasar hasta llegar al objeto deseado, separados por puntos. De esta forma, si queremos acceder al objeto *mib-2* de la figura 2.1, el OID para ello será *1.3.6.1.2.1.*

Tipo de dato	**Descripción**
INTEGER	Entero de 32 bits. En SMIv2 pasó a llamarse *Integer32*.
OCTECT STRING	Cadena de bytes utilizada para cadenas de texto y direcciones.
Counter	Número de 32 bits cuyo mínimo valor es 0 y su máximo valor es (2^{32} - 1). Cuando alcanza su valor máximo, vuelve a 0. Se utiliza para contabilizar eventos como el número de bytes enviados o el número de errores detectado. Al ser utilizado como contador, su valor nunca decrece. En SMIv2 pasó a llamarse *Counter32*
Counter64	Tipo de dato introducido en SMIv2, que es igual a *Counter*, pero con una mayor rango, ya que en este caso son 64 bits en vez de 32.
OBJECT IDENTIFIER	Lista de números decimales separados por puntos que representa un elemento dentro del árbol MIB.
NULL	No se utiliza en SNMP.
SEQUENCE	Lista que contiene cero o más tipos de datos ASN.1.
SEQUENCE OF	Define un objeto gestionado que ha sido creado a partir de un *SEQUENCE*.
IpAddress	Representa una dirección IP de 32 bits. No se incluyen direcciones IPv6.
NetworkAddress	Dato con las mismas características que *IpAddress*, pero que sirve para representar otro tipo de direcciones, no exclusivamente direcciones IP.
Gauge	Número positivo de 32 bits que, a diferencia de *Counter*, puede decrementar su valor.En SMIv2 pasó a llamarse *Gauge32*.
TimeTicks	Número positivo de 32 bits que se emplea para medir tiempo en centésimas de segundo.
Opaque	Tipo de dato que permite a cualquier otra codificación ASN.1 ser almacenada como un *OCTECT STRING*.
Unsigned32	Tipo de dato introducido en SMIv2 que representa valores decimales entre 0 y 2^{32} - 1, ambos incluidos.
BITS	Enumeración de bits.

Tabla 2.1: Tipos de datos definidos por la SMI [Mau05, McC88b]

.

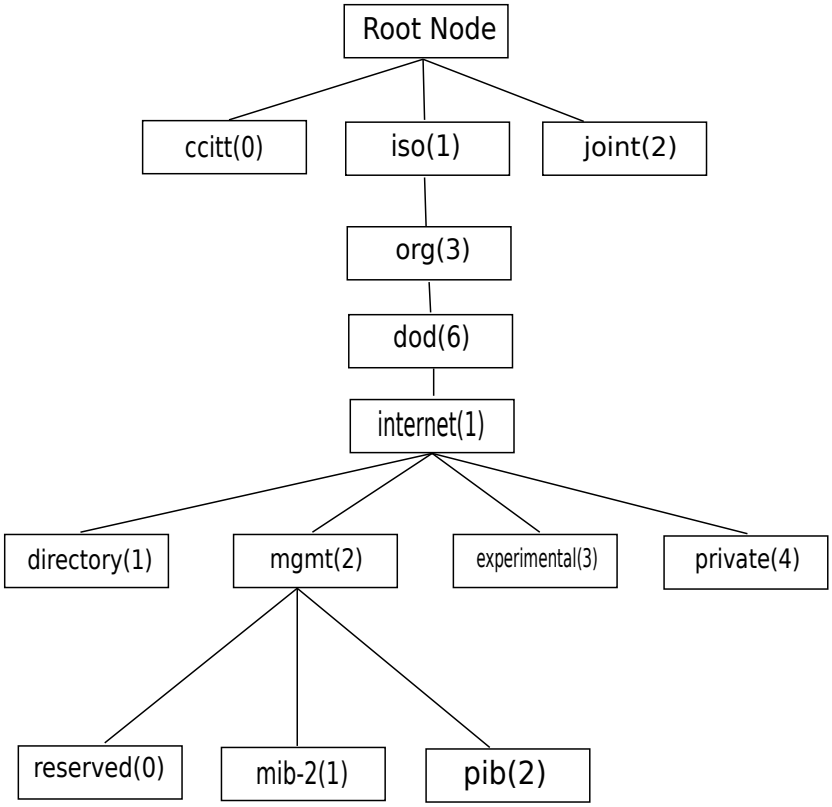


Figura 2.1: Árbol SMI.

Operaciones SNMP

SNMP incluye tres tipos de operaciones con las que un NMS puede realizar solicitudes a los agentes SNMP. Además incluye una operación conocida como trampa, con la que el agente notifica al servidor SNMP en caso de que se produzca un cambio de estado relevante en algún elemento del árbol.

En primer lugar vamos a definir las operaciones utilizadas por el NMS para realizar las peticiones a los agentes SNMP.

set Esta operación se utiliza para cambiar el valor de un objeto del árbol SMI del agente al que se realiza la petición. Hay algunos parámetros que no pueden ser modificados, ya que son de solo lectura y otros pueden ser configurados para permitir o no la modificación. En un escenario de éxito, el servidor SNMP mandaría la petición de modificación, el agente SNMP modificaría el dato y una vez

```
dot1dTpFdbStatus OBJECT-TYPE
    SYNTAX      INTEGER {
                    other(1),
                    invalid(2),
                    learned(3),
                    self(4),
                    mgmt(5)
                }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The status of this entry.  The meanings of the
        values are:
            other(1) - none of the following.  This would
                include the case where some other MIB object
                (not the corresponding instance of
                dot1dTpFdbPort, nor an entry in the
                dot1dStaticTable) is being used to determine
                if and how frames addressed to the value of
                the corresponding instance of
                dot1dTpFdbAddress are beeing forwarded.
            invalid(2) - this entry is no longer valid (e.g.
                it was learned but has since aged out), but
                has not yet been flushed from the table.
            learned(3) - the value of the corresponding
            instance of dot1dTpFdbPort was learned, and is
            being used.
            self(4) - the value of the corresponding
            instance of dot1dTpFdbAddress represents one of
            the bridge's addresses.  The corresponding
            instance of dot1dTpFdbPort indicates which of
            the bridge's ports has this address.
            mgmt(5) - the value of the corresponding
            instance of dot1dTpFdbAddress is also the value
             of an existing instance of dot1dStaticAddress."
    ::= { dot1dTpFdbEntry 3 }
```

Figura 2.2: Definición del objeto dot1qTpFdbPort.

actualizado el agente notificaría al NMS del cambio realizado, como se puede ver en la figura 2.3. De ocurrir algún error, el agente SNMP contestaría con un código de error explicando el motivo del error. En la tabla 2.2 se recogen los distintos códigos de error, donde los 5 primeros fueron introducidos en SNMPv1, mientras que el resto se introdujeron en la segunda versión del protocolo [Cas88, Pri88].

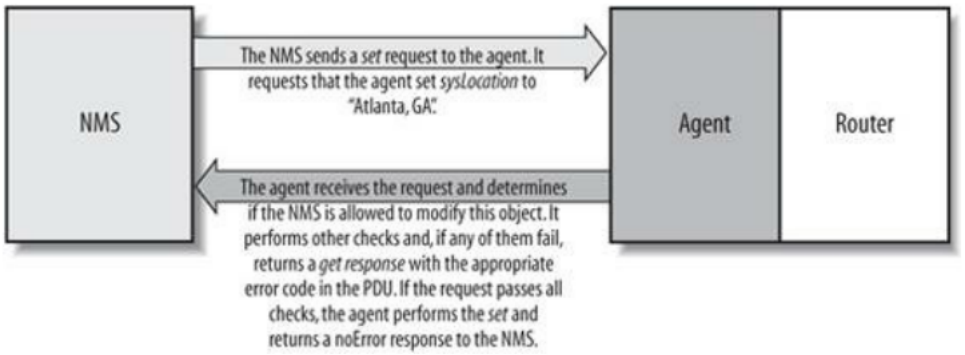


Figura 2.3: Ejemplo de intercambio de mensajes una operación set. [Mau05]

get La operación `get` sirve para consultar el estado de un objeto del árbol. Al igual que en la operación *set*, el objeto viene dado por su OID y la operación la inicia siempre el NMS, que es quien envía la petición al agente SNMP. En la figura 2.4 se puede observar el intercambio de mensajes entre servidor y agente en una petición get.

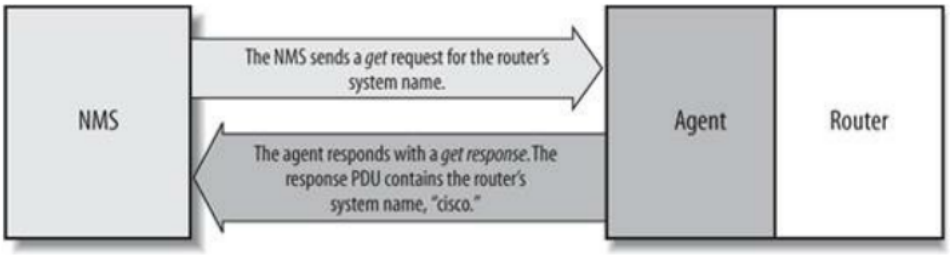


Figura 2.4: Ejemplo de intercambio de mensajes una operación get [Mau05].

getnext Esta operación permite realizar una serie de operaciones `get` con el objetivo de obtener como respuesta un subárbol completo. Esto es posible debido

Código	**Descripción**
noError(0)	La operación se ha realizado exitosamente.
tooBig(1)	La respuesta a la petición es demasiado grande y no puede ser enviada en un único mensaje
noSuchName(2)	No se ha encontrado el OID proporcionado.
badValue(3)	El valor proporcionado en el set no es del mismo tipo que el del valor que se intenta modificar.
readOnly(4)	El objeto que se intenta modificar solo tiene acceso de lectura.
genErr(5)	Si el error ocurrido no puede englobarse en ninguno de los casos previos. Hay que tener en cuenta que este código es de la primera versión del protocolo. En su segunda versión se añadirían 13 códigos de error más.
noAccess(6)	El objeto no es accesible, al menos para ese NMS.
wrongType(7)	El valor proporcionado en el set no es del mismo tipo que el del valor que se intenta modificar. Funciona igual que *badValue*.
wrongLength(8)	Se ha proporcionado un valor en el set de dimensiones distintas a las esperadas.
wrongEncoding(9)	La codificación del valor proporcionado es errónea.
wrongValue(10)	Se ha proporcionado un valor incorrecto. Este código sería devuelto, por ejemplo, si intentamos cambiar el valor del objeto definido en la figura 2.2 por uno que no estuviera entre uno y cinco.
noCreation(11)	Se ha intentado asignar un valor a un objeto que no existe o no ha sido creado en el árbol MIB.
inconsistentValue(12)	La variable está en un estado inconsistente, y por ello no acepta modificar su valor.
resourceUnavailable(13)	La asignación del valor requiere la utilización de un recurso no disponible en el momento que ha recibido la operación set.
commitFailed(14)	Se origina si el error no puede ser englobado en ningún otro código de error.
undoFailed(15)	Se devuelve si ha ocurrido un error en el proceso de cambiar el estado de un objeto y no se puede recuperar su estado original.
authorizationError(16)	Ha ocurrido un error de autorización.
notWritable(17)	Indica que la variable no puede ser modificada.
inconsistentName(18)	Indica que la variable a actualizar estaba se encuentra en un estado inconsistente y por ello no se puede realizar la operación.

Tabla 2.2: Códigos de error SNMP

a la distribución en forma de árbol de los objetos, ya que el agente es capaz de rastrear el objeto desde el nodo principal, y así devolver el subárbol al completo. Cuando un NMS recibe una respuesta a un comando *getnext*, emite otro comando, y así hasta recibir un error, que significa que se ha alcanzado el final del subárbol de la MIB y ya no quedan objetos que obtener. [Mau05]

getbulk Se trata de una operación incluida en la segunda versión del protocolo cuya utilidad radica en la obtención de una parte del árbol en una sola solicitud, sin necesidad de realizar una serie de peticiones `getnext` concatenadas. Si el agente no puede devolver todos los datos solicitados en una sola respuesta, responde con un mensaje de error sin datos. La operación puede solicitar al agente que devuelva la mayor cantidad de datos posibles, por lo que es posible que esto origine respuestas incompletas debido a la incapacidad de incluir todos los datos en un mismo mensaje de respuesta. En la figura 2.5 se puede ver la secuencia de mensajes enviados en una petición *getbulk*.

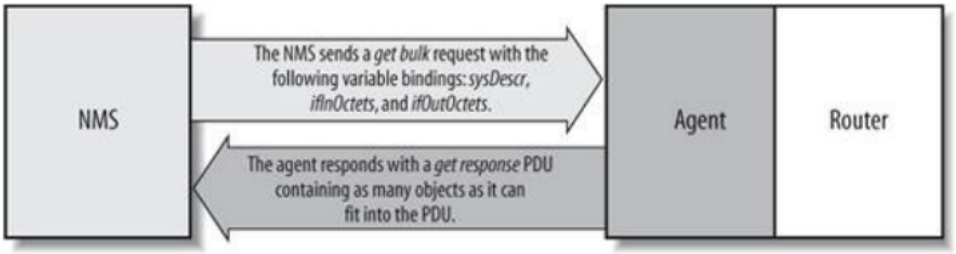


Figura 2.5: Ejemplo de intercambio de mensajes de una operación getbulk [Mau05].

Para terminar con las operaciones del protocolo SNMP, vamos a hablar en el siguiente apartado de las trampas, que van a resultar fundamentales en el desarrollo de la aplicación.

Trampas SNMP

Las trampas SNMP, renombradas en la versión 2 del protocolo como notificaciones, son la manera que tiene un agente SNMP de comunicarse con el servidor para notificarle que algo ha ocurrido. Para el desarrollo este TFG resultarán fundamentales, puesto que es necesario saber cuando se conecta o se desconecta un dispositivo del *switch*.

Cuando un agente SNMP detecta un evento que deba ser notificado, el agente envía la notificación al destinatario o destinatarios de la trampa, algo que debe haberse configurado previamente. Además, al usar UDP, el agente no recibe ningún acuse de recibo de que la trampa ha llegado al servidor, por lo que es posible que la trampa se pierda y el NMS no quede notificado [Sta98]. La figura 2.6 muestra un ejemplo del envío de una trampa por parte de un agente SNMP.

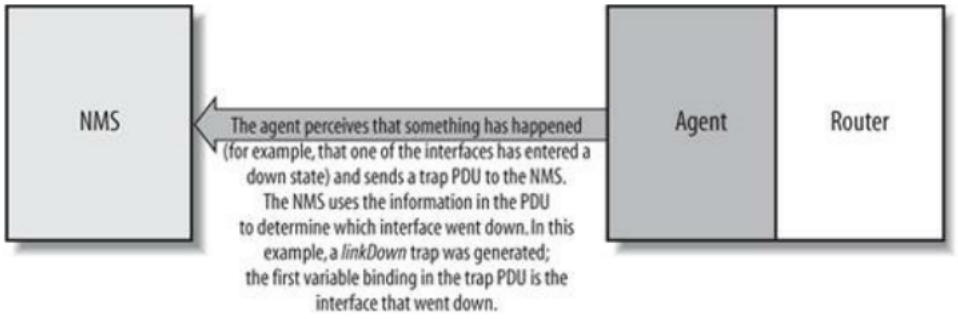


Figura 2.6: Ejemplo de envío de trampa [Mau05].

Cuando un NMS recibe una trampa, debe saber como interpretarla y cuál es su contenido. Por ello, las trampas se identifican mediante un número de trampa. Hay 7 tipos de trampas diferentes, que están definidas en la tabla 2.3, aunque los fabricantes pueden crear sus propias notificaciones, para lo que se reserva el identificador de trampa 6. Además, la propia trampa contiene cierta información como el valor de los OID a los que afecta [Cas88].

Código	**Descripción**
coldStart(0)	Indica que el agente SNMP ha sido reiniciado. Pone a 0 todas las variables de gestión, como los contadores. Puede ser utilizada para conocer que un nuevo dispositivo ha sido añadido a la red.
warmStart(1)	Indica que el agente ha sido reiniciado por él mismo y ninguna de sus variables de gestión es reiniciada.
linkDown(2)	Indica que una interfaz del dispositivo gestionado se ha desactivado. El primer dato contenido en la trampa identifica la interfaz que ha sido desactivada.
linkUp(3)	Indica que una interfaz de un dispositivo gestionado ha sido activada. El primer datos contenido en la trampa identifica la interfaz que ha sido activada.

Código	**Descripción**
authenticationFailure(4)	Indica que se ha intentado realizar una consulta al agente con una comunidad errónea. Es útil para determinar si alguien está intentando acceder a alguno de los dispositivos gestionados.
egpNeighborLoss(5)	Indica que se ha perdido un vecino EGP *(Exterior Gateway Protocol)*.
enterpriseSpecific(6)	Indica que se trata de una trampa específica del fabricante.

Tabla 2.3: Tipos de trampa SNMP [Cas90].

2.2. VLAN

El segundo componente clave del presente Trabajo de Fin de Grado son las **VLAN** *(Virtual Local Area Network)*, ya que gracias a ellas se podrá hacer la división lógica de los usuarios en redes distintas, en función del rol que tengan dentro de la red, para así poder aplicar las políticas de acceso y seguridad que el administrador de la red considere oportunas.

2.2.1. Origen

En un principio, una red de área local o LAN *(Local Area Network)*, estaba definida como una red de ordenadores que compartían un mismo segmento de red. Esto hacía que las redes fueran poco escalables, debido a la complejidad y coste que conllevaba su crecimiento. Cuando mayor fuera la red, mayor sería el número de equipos capaces de enviar mensajes de difusión, con lo que conforme la red fuera creciendo, más sobrecarga habría. Además, la topología utilizada por excelencia, ya que era la que más velocidad de transmisión proporcionaba y más fácil de implantar resultaba, era el bus. Esto acrecentaba los problemas debido a las restricciones que impone CSMA/CD *(Carrier Sense Multiple Access with Collision Detection)*, el protocolo de acceso a medio utilizado, en términos de tamaño de la trama y longitud del segmento de cable. [Ins]

Para evitar este problema, la única solución que había era la colocación de un *router* para separar dominios de difusión y evitar el tránsito masivo de tramas de difusión. Esta solución tenía sus inconvenientes, principalmente dos: el elevado

coste, por normal general, del *router* a incluir en la red y el retardo que impone en la red, puesto que generalmente un *router* introduce un retardo mayor al de un *switch* o un *hub*, ya que tiene que procesar los paquetes antes de reenviarlos [Sta06].

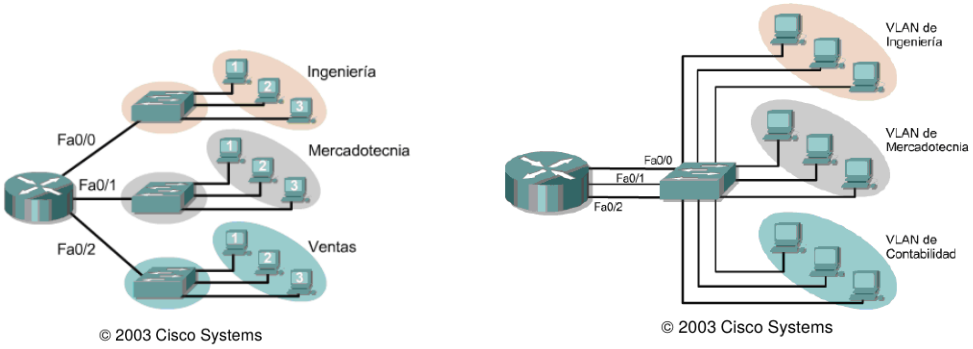
Esta solución, con sus pros y sus contras, seguía quedándose corta, puesto que la red seguía estando definida por el segmento físico donde se encontraban los elementos que la componían, cuando lo deseable sería que una red estuviera definida por su dominio de difusión, independientemente del segmento de red en el que se encuentre, pudiendo ser este compartido por otras redes.

Por lo comentado anteriormente, W. David Sincoskie ideó la creación de las las redes virtuales, con la intención de dar una solución definitiva al problema y evitar la utilización de routers para separar redes de difusión [Sin02].

2.2.2. Definición y características

Una red virtual de área local es una agrupación lógica de dispositivos de red que conforman un mismo dominio de difusión. Estos dispositivos pueden estar en la misma LAN o en diferentes redes físicas, así como puede haber varias VLAN en un mismo segmento físico. De esta forma, es posible crear una red de ordenadores siguiendo criterios lógicos en lugar de físicos. A diferencia de las redes tradicionales definidas por un *router*, en este caso es un switch el que se encarga de realizar la separación de redes. De esta forma, los mensajes de difusión que envíe un usuario de una VLAN solo serán reenviados por aquellos puertos del *switch* que pertenezcan a la VLAN. Esto no hace que los *routers* sean prescindibles, ya que los *host* que se encuentran vinculados a una VLAN solo pueden comunicarse con los *host* de su misma VLAN, siendo necesario un router para la comunicación entre diferentes VLAN. En las figuras 2.7(a) y 2.7(b) se puede ver la diferencia entre una red tradicional y una red que utiliza VLAN [Lam06].

Al implantar VLAN se solucionan muchos de los problemas ocasionados por un switch de capa dos. Como ya se ha comentado, uno de ellos es la disminución del tamaño de los dominios de difusión, puesto que se aumenta el número de los mismos. En el aspecto de la seguridad también se pueden encontrar grandes beneficios ya que, si hay un solo dominio de difusión, todos los dispositivos son visibles entre sí, y esto puede ser, en ocasiones, algo no deseado. En cambio, creando redes virtuales, se puede evitar que esto ocurra cambiando a los usuarios de red en función de las necesidades del administrador. He aquí algunas ventajas con respecto a la tradicional conmutación de nivel 2 OSI [Lam06]:



(a) Agrupamiento lógico tradicional. (b) Agrupamiento lógico con VLAN.

Figura 2.7: Agrupamiento lógico con VLAN

- En primer lugar, motivo por el cual las VLAN son tan importantes para el desarrollo de la aplicación, es posible crear una red restringida para los usuarios no autenticados de la red. De esta forma es posible aislarlos del resto de componentes y recursos de la red simplemente con el hecho de asociarlos a una VLAN determinada.

- De la misma manera que en el caso anterior, es posible aislar a los elementos y usuarios de la red que necesiten permisos especiales, y así nadie ajeno a esa VLAN podrá comunicarse con ellos.

- Es posible agrupar a los diferentes usuarios de la red en subredes en función de su rol, aunque compartan el mismo segmento físico. Esto es útil para dividir a los usuarios en función de los departamentos en los que trabajen dentro de una empresa o para separar a los alumnos de lo profesores en el ámbito educativo.

- Cambiar a un usuario de red es tan fácil como modificar la VLAN del puerto al que está conectado.

- Facilita la tarea del administrador a la hora de aplicar reglas en un corta-fuegos, puesto que estas reglas podrán ser aplicadas en función de la red, si cada usuario de la red está en la VLAN que le corresponde.

- El *switch* solo puede ser configurado a través de la denominada VLAN de gestión, evitando así que cualquiera pueda entrar en la configuración del *switch* y alterar su configuración.

Una vez definido el concepto de VLAN y explicado algunas de sus ventajas frente a las redes físicas, es necesario continuar explicando cómo se crea una VLAN y que tipos hay.

Como se ha mencionado anteriormente, las redes virtuales se crean en un *switch*. Cada puerto del switch tiene asociada una VLAN, y de esta forma, cuando le llega un mensaje de difusión de una determinada red, sabe por que puertos tiene que reenviarlo y por cuales no. Una misma VLAN puede estar presente en más de un switch, mientras que un puerto puede tener asociada más de una VLAN. Hay dos tipos distintos de VLAN:

- **VLAN estática.** También se conoce cómo VLAN basada en puerto. Cada puerto del *switch* tiene asociado un único identificador de VLAN por defecto y sea cual sea el host conectado al ese puerto estará siempre en la misma red, salvo que se trate de un puerto troncal. Son configuradas de forma manual por el administrador. En este caso, las tramas emitidas por los *host* de la VLAN no tienen la obligación de ir etiquetadas, concepto que explicaremos a continuación.
- **VLAN dinámica.** Pueden estar basadas en parámetros como la dirección MAC del *host* o el protocolo utilizado. Requiere una menor supervisión por parte del administrador. Para el caso de las VLAN basadas en dirección MAC, cuando un usuario se conecta a la red se le asigna a una red u otra en función de la dirección. Este tipo de VLAN es sensible a recibir ataques del tipo MAC spoofing, donde un *host* cambia su dirección MAC, pudiendo entrar de esta forma a una red que no le corresponde.

Como se ha comentado anteriormente, los *hosts* de una VLAN solo son visibles dentro de esa VLAN. Si se tienen varios *switches* con las mismas VLAN configuradas (póngase el caso de un edificio con un *switch* por planta) y se quiere conectarlos, habrá que establecer un enlace entre los *switches* por cada VLAN que haya. Si el número de VLAN presentes es pequeño, puede ser algo asumible. Pero ¿qué ocurre si hay un número elevado de redes? Es inviable mantener un enlace entre los *switches* por cada VLAN que haya. Para dar solución a este problema aparece el estándar 802.1q, que define los enlaces *trunk* o troncales.

Un enlace troncal sirve para transportar tráfico de varias VLANs entre dos *switches* o entre un *swtich* y un *router*. Para distinguir el tráfico de las diferentes VLANs que viaja por estos enlaces, las tramas son etiquetadas con el VID *(VLAN IDentifier)* de la red. Esta etiqueta consiste en un campo de 12 bits que se añade a la trama Ethernet, y que pasa a denominarse trama 802.1q. Es el *switch* el que

se encarga de etiquetar las tramas antes de enviarlas por el enlace troncal. Cuando recibe una trama por un puerto de este tipo, mira cual es su etiqueta y la envía por un puerto troncal (sin modificarla) o quita la etiqueta y la envía por un puerto de acceso, lo cual hace que distingamos entre tres tipos de puertos en un switch en el que se implementan VLANs [Kee00].

- **Puerto troncal:** es un puerto que tiene asignada más de una VLAN, por lo que para enviar una trama por dicho puerto hay que añadir una etiqueta para identificar a que red pertenece. En el switch, todos las VLAN que pertenecen al puerto se definen como *Tagged* (etiquetados).
- **Puerto de acceso:** se trata de un puerto que únicamente tiene una VLAN asociada. Por este puerto las tramas circulan sin etiquetar. Un host conectado a uno de estos puertos desconoce que forma parte de una VLAN. El puerto cuenta con una única VLAN configurada como *Untagged* (no etiquetados).
- **Puerto híbrido:** es un puerto no contemplado en ninguno de los dos casos anteriores. Tiene una VLAN configurada como *Untagged* y el resto configuradas como *Tagged*.

Capítulo 3

Análisis y diseño

Para llevar a cabo la elaboración del presente trabajo se ha seguido un proceso de ingeniería de *software*. La ingeniería de *software* es una disciplina dentro del área de la informática o computación que ofrece métodos, técnicas y procedimientos para desarrollar y mantener un *software* de calidad. En [IEE90] se define la ingeniería de software como «la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento *software*». De esta forma, el desarrollo de un componente software está dividido en varias fases. Estas son: comunicación, análisis, modelado, construcción, despliegue y mantenimiento.

En le presente capítulo se lleva a cabo el desarrollo de la segunda fase, que comprende el análisis y el diseño de la aplicación [Pre10].

3.1. Planteamiento

Antes de comenzar con el análisis de los requisitos que debe cumplir la aplicación, es necesario realizar una descripción general del funcionamiento de la herramienta. La idea es construir una herramienta que proporcione control de acceso en una red donde los usuarios acuden con su propio equipo y se conectan. Para poder tener acceso a la red es necesario que estén dados de alta en el sistema y autenticarse con sus credenciales. Mientras el dispositivo conectado no esté autenticado, permanecerá en una VLAN restringida, sin acceso a red. Además de esta VLAN, habrá otras dos redes más, una donde se encuentran los usuarios que se han autenticado, y otra VLAN donde permanecen los dispositivos cuya dirección

MAC es conocida y que no necesitan pasar por el proceso de autenticación, puesto que se conectan desde un dispositivo conocido por el sistema.

Para proporcionar un mecanismo de autenticación a los usuarios, será necesario poner en marcha un servidor web que aloje una página web que solicite las credenciales al usuario y, en caso de ser correctas, lo cambie a la VLAN de usuarios autenticados. Un elemento dentro de la red que no puede faltar es un servidor DHCP que proporcione los parámetros de red necesarios a los usuarios conectados a la red. Además, será necesario también un servidor DNS que atienda las consultas de traducción de nombre de dominio requeridas por los usuarios. Este servidor debe estar configurado de tal forma que las consultas realizadas por usuarios que se encuentren en la VLAN restringida sean respondidas con la dirección IP del servidor que contiene la página web de acceso al sistema. De esta forma se facilita el acceso a los usuarios.

Todo este conjunto de elementos conforman un mecanismo de control de acceso a red que separa a los usuarios de la red en tres redes distintas, en función de su rol, facilitando así la aplicación de políticas de cortafuegos. Además del control de acceso, se pretende dotar al sistema de una herramienta gráfica que muestre cual es el estado actual de los puertos del switch (conectado o no, usuario conectado, red en la que se encuentra, etc). Para realizar esto será necesario incorporar una base de datos a la aplicación que contenga los datos más relevantes del switch, además de contener los usuarios que están dados de alta en el sistema.

3.2. Análisis

Dentro del contexto de la ingeniería de software, la fase de análisis comprende el estudio y recopilación de las características deseadas para la aplicación a desarrollar, así como de los requisitos que debe de cumplir y las situaciones que debe ser capaz de abordar el programa.

El análisis puede comprender diferentes fases, aunque para la elaboración del presente Trabajo de Fin de Grado nos hemos centrado en las dos más importantes: el análisis de los requisitos del sistema y especificación de casos de uso.

3.2.1. Requisitos del sistema

En el ámbito de la ingeniería de *software*, un requisito del sistema se define como una condición o capacidad que necesita un sistema *software* para resolver un problema determinado y alcanzar sus objetivos [IEE90]. Este concepto también es aplicado a las condiciones que debe poseer un sistema para satisfacer un contrato o especificación.

Se suelen definir dos tipos de requisitos: funcionales y no funcionales. En ocasiones se denomina a los requisitos no funcionales como requisitos **FURPS**, nombre que deriva del acrónimo en ingles de las categorías que lo componen *(Functionality,Usability, Reliability, Performance, Supportability)*. Para hacer más sencillo el análisis de requisitos, se ha decidido no separar los requisitos no funcionales en requisitos FURPS.

Requisitos funcionales

Un requisito funcional del sistema se define como una característica que expresa una funcionalidad del programa y que puede ser satisfecho mediante la adición de un bloque de código en el *software*. Se han identificado un total de diez requisitos funcionales, que se muestran en la tabla 3.1 [Som10].

Identificador	**Descripción**
FRQ-001	El sistema deberá ser capaz de mostrar el estado actual de los puertos del *switch*.
FRQ-002	El sistema deberá permitir autenticarse a los clientes de la red para poder acceder.
FRQ-003	El sistema deberá gestionar la conexión de un puerto del *switch*.
FRQ-004	El sistema deberá gestionar la desconexión de un puerto del *switch*.
FRQ-005	El sistema deberá ser capaz de cambiar la VLAN asociada a un puerto del *switch*.
FRQ-006	El sistema deberá ser capaz de identificar los hosts conocidos por el sistema cuando se conecten al *switch*.
FRQ-007	La interfaz gráfica deberá actualizarse cuando el estado del *switch* cambie.
FRQ-008	El sistema deberá permitir al administrador cambiar los parámetros relativos al *switch*, como dirección IP y número de puertos, de la base de datos.

Identificador	Descripción
FRQ-009	El sistema deberá contar con un mecanismo para actualizar la base de datos obteniendo la información directamente del *switch*. De esta forma podrá disponer de información veraz a la hora de poner en marcha la aplicación.
FRQ-010	El sistema deberá permitir ver el estado de los puertos del *switch* de forma detallada.

Tabla 3.1: Requisitos funcionales.

Requisitos no funcionales

Una vez enumerados los requisitos funcionales de nuestro sistema, es necesario hacer lo propio con los requisitos no funcionales del mismo. Un requisito no funcional se define como aquella característica del sistema que, sin suponer necesariamente una funcionalidad, supone una restricción del sistema. Es decir, definen propiedades emergentes del sistema, como el tiempo de respuesta o el sistema *software* a utilizar. Este tipo de requisitos no tiene por qué ser abordados mediante la adición de *software* necesariamente. Se han encontrado un total de tres requisitos no funcionales, como se puede ver en la tabla 3.2.

Identificador	Descripción
NFRQ-001	El sistema deberá permitir refrescar la interfaz gráfica.
NFRQ-002	La versión de SNMP a utilizar será la segunda, en su variante basada en comunidades.
NFRQ-003	La información mostrada en la ventana principal debe ser simple y directa. Así como la navegación por los menús debe ser intuitiva.

Tabla 3.2: Requisitos no funcionales.

3.2.2. Casos de uso

Una vez abordado el análisis de los requisitos del sistema es necesario continuar con la fase de análisis de nuestro proyecto con la enumeración de los diferentes casos de uso encontrados. En primer lugar, es necesario definir qué es un caso de uso. Dentro de la ingeniería de *software*, un caso de uso es la descripción del comportamiento del sistema *software* o, en su defecto, de cómo se desea que sea su

comportamiento y forma de funcionar. En las tablas 3.3 a 3.9 se definen, de forma textual, los casos de uso para esta herramienta.

ID	LinkUp
Satisface	FRQ-003, FRQ-005, FRQ-006 y FRQ-007
Actores implicados	Base de datos, servidor SNMP y *switch*
Prerrequisitos	Un puerto del *switch* ha sido activado, ya sea por la conexión del un nuevo dispositivo al puerto o por la activación del dispositivo conectado a él.
Escenario principal de éxito o flujo normal	
1	El demonio *snmptrapd* captura la trampa enviada por el agente SNMP y realiza la llamada al *script* encargado del la gestión del *switch*, pasando como argumento el tipo de evento producido y el puerto del *switch*.
2	El servidor obtiene la dirección MAC del dispositivo que se acaba de conectar al *switch* mediante SNMP.
3	Se comprueba si la trampa se se ha producido por un cambio de estado del puerto del *switch* provocado por la aplicación.
4	Se comprueba si la dirección MAC es conocida. Si lo es cambia el puerto a la VLAN correspondiente.
5	Se cambia el estado del puerto implicado en la base de datos.
6	Se envía una señal para que la GUI actualice su estado.
Postcondición	El dispositivo conectado se guarda en la base de datos, se le asigna la VLAN correspondiente y el cambio se ve reflejado en la GUI.
Flujo alternativo	
3	Si la trampa no ha sido producida por la conexión de un nuevo dispositivo al *switch*, finaliza la tarea y cambia el campo 'puerto_manipulado' de la base de datos a 0.

Tabla 3.3: Caso de uso LinkUp.

ID	LinkDown
Satisface	FRQ-004, FRQ-005 y FRQ-007
Elementos que intervienen	Base de datos, servidor SNMP y switch
Prerrequisitos	Un puerto del *switch* ha sido desconectado, ya sea por la desconexión de un dispositivo conectado a ese puerto o por el apagado del mismo.
Escenario principal de éxito o flujo normal	

1	El demonio *snmptrapd* captura la trampa enviada por el agente SNMP y realiza la llamada al script encargado del la gestión del *switch*, pasando como argumento el tipo de evento producido y el puerto del *switch*.
2	Se comprueba si la trampa se se ha producido por un cambio de estado del puerto del *switch* provocado por la aplicación.
3	Se obtiene la VLAN en la que se encuentra el puerto y lo cambia, si fuese necesario, a la VLAN restringida.
4	Se cambia el estado del puerto implicado en la base de datos.
5	Se envía una señal para que la GUI actualice su estado.
Postcondición	El dispositivo desconectado desaparece de la base de datos, el puerto vuelve a tener asignada la VLAN de usuarios restringidos y se actualiza la GUI para reflejar el cambio.
Flujo alternativo	
2	Si la trampa no se ha producido por la desconexión de un dispositivo del *switch*, finaliza la tarea.

Tabla 3.4: Caso de uso LinkDown.

ID	CambiaValores
Satisface	FRQ-008
Elementos que intervienen	Administrador, GUI y base de datos
Prerrequisitos	En el menú de la aplicación el administrador ha pulsado el botón "Gestionar Switch".
Escenario principal de éxito o flujo normal	
1	Se abre una ventana con los parámetros que es posible modificar.
2	Se realiza una consulta a la base de datos para obtener el valor actual de los parámetros.
3	Se muestra el valor actual de los parámetros a modificar.
4	El administrador modifica los parámetros del *switch* que desee cambiar en la base de datos.
5	El administrador pulsa el botón 'Guardar' y se despliega una ventana emergente notificando el cambio.
Postcondición	Los cambios introducidos quedan guardados en la base de datos.
Flujo alternativo	
5	El administrador pulsa el botón 'Cancelar' para volver a dejar el valor guardado y sale.

Tabla 3.5: Caso de uso CambiaValores.

ID		InicializaBD
Satisface		FRQ-009
Elementos que intervienen		Administrador, GUI, switch y base de datos
Prerrequisitos		En el menú de la aplicación el administrador ha pulsado el botón 'Actulizar BD'.
Escenario principal de éxito o flujo normal		
1	Mediante operaciones `get` SNMP se obtiene el estado actual de los puertos del *switch* y se compara con el de la base de datos. En caso de no tener el mismo valor que el obtenido se cambia.	
2	Se obtiene la dirección MAC del dispositivo conectado a cada puerto, si es que lo hubiera, y se coteja con la que consta en la base de datos. Si difiere del resultado obtenido se cambia.	
3	Se obtiene el identificador de VLAN asociado a cada puerto y se compara con el que consta en la base de datos. Si difiere del resultado obtenido se cambia.	
4	Para finalizar, se muestra una ventana emergente indicando si la base de datos estaba actualizada o no	
Postcondición		La tabla de la base de datos que contiene el estado del *switch* queda actualizada a partir de los datos obtenidos del switch.

Tabla 3.6: Caso de uso InicializaBD.

ID		MuestraDetallePuertos
Satisface		FRQ-010 y FRQ-001
Elementos que intervienen		Administrador, GUI y base de datos
Prerrequisitos		En el menú de la aplicación el administrador ha pulsado el botón 'Ver Puertos'.
Escenario principal de éxito o flujo normal		
1	Se realiza una consulta a la base de datos para obtener los datos de la tabla que contiene el estado de los puertos del *switch*, así como la dirección MAC asociada y el usuario conectado a cada puerto, si es que lo hubiera.	
2	Se muestra la información obtenida anteriormente en formato de tabla.	

Postcondición	Se muestra en una nueva ventana una tabla que, para cada puerto, contiene los siguientes datos: Estado (conectado o desconectado), dirección MAC del dispositivo conectado, login del usuario conectado a dicho puerto y el identificador de VLAN del puerto.

Tabla 3.7: Caso de uso MuestraDetallePuertos.

ID	AccesoSistema
Satisface	FRQ-002 y FRQ-007
Elementos que intervienen	Servidor SNMP, switch y base de datos
Prerrequisitos	Un usuario se ha conectado al *switch* y al realizar una búsqueda en Internet ha sido redirigido a la página web de acceso al sistema.
Escenario principal de éxito o flujo normal	
1	El usuario introduce su usuario y contraseña para acceder a la red.
2	Se comprueba si existe en la base de datos el par usuario-contraseña introducido por el cliente.
3	A partir de la dirección IP del usuario se obtiene la dirección MAC del dispositivo
4	A partir de la dirección MAC del dispositivo se obtiene el puerto del switch al que está conectado
5	Se cambia el identificador de VLAN asociado al puerto al que está conectado el usuario al correspondiente a la VLAN de usuarios autenticados
6	Se cambia el identificador de VLAN del puerto de la base de datos y se añade el usuario que se acaba de conectar a la tabla correspondiente.
7	Envía una señal para que la GUI actualice su estado.
Postcondición	El usuario se autentica con éxito y es cambiado de VLAN para que pueda tener conexión. Se actualiza la base de datos con los cambios acontecidos y se actualiza la interfaz.
Flujo alternativo	
2	El par usuario-contraseña no figura en la base de datos, por lo que se vuelve a pedir al usuario que vuelva a introducir sus credenciales.

Tabla 3.8: Caso de uso AccesoSistema.

ID	ActualizaInterfaz
Satisface	FRQ-007 y NRFQ-001
Elementos que intervienen	Servidor SNMP, administrador y base de datos
Prerrequisitos	Se ha solicitado la actualización o refresco de la ventana principal de la aplicación. Esto puede deberse a dos motivos: se ha producido un cambio en alguno de los puertos del *switch* o el administrador a pulsado el botón 'Actualizar interfaz' del menú de la ventana principal.
Escenario principal de éxito o flujo normal	
1	Se realiza la consulta a la base de datos para obtener los datos relativos al estado del *switch*.
2	En función del estado de cada puerto, se colorea de un color u otro la línea que une el puerto del *switch* con el icono del ordenador en la ventana principal.
3	Se rellena el cuadro de texto de cada pc conectado al *switch* con la información más relevante. Es decir, estado, usuario y dirección MAC.
Postcondición	La ventana principal de la aplicación muestra el estado actual del *switch*, mostrando para cada puerto la información más relevante.

Tabla 3.9: Caso de uso ActualizaInterfaz.

Una vez expuestos de manera textual todos los casos de uso de nuestra aplicación, es necesario mostrar como se relacionan entre sí. Para ello vamos a hacer uso del **diagrama de casos de uso**, para así dar por finalizada la fase de análisis. En este diagrama aparecerán los casos de uso mencionados anteriormente y los actores con los que interactúan, que son dos: el administrador y el usuario conectado a la red. En la figura 3.1 se puede observar como quedaría el diagrama de casos de uso.

3.3. Diseño

Una vez terminada la fase de análisis de requisitos y especificación de casos de uso de la aplicación, el siguiente paso es realizar el diseño. Para ello se ha seguido el patrón MVC *(Modelo-Vista-Controlador)*. Este patrón de arquitectura *software* consiste en separar los datos (modelo) de la aplicación, de la interfaz de usuario (vista) y de la lógica de control (controlador), contando así con tres componentes

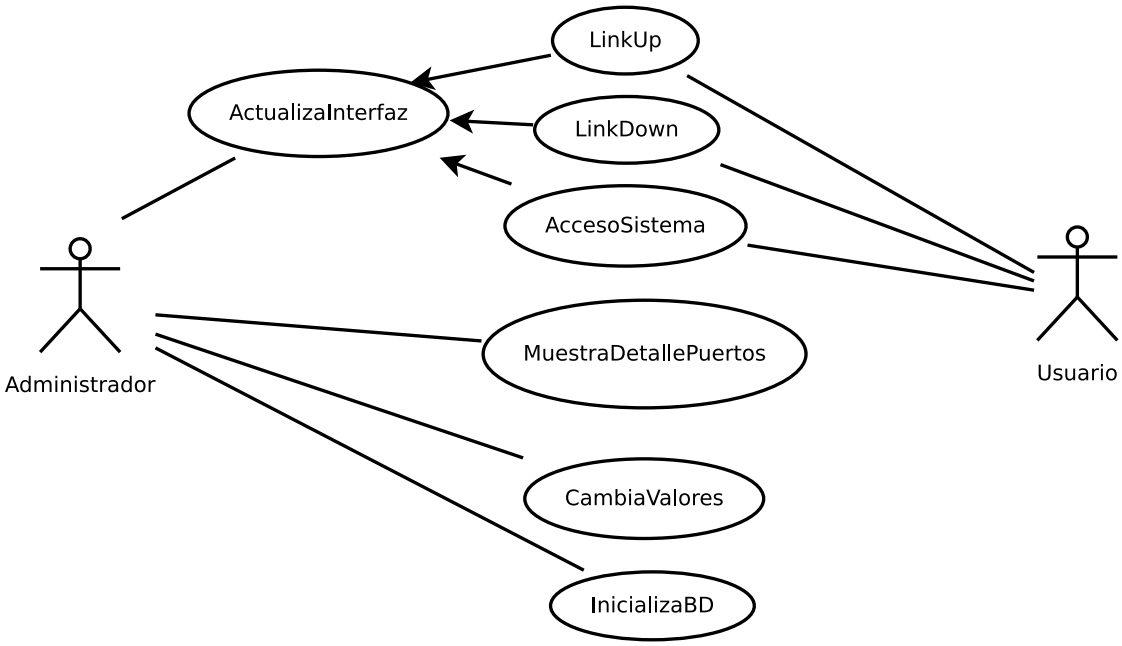


Figura 3.1: Diagrama de casos de uso.

bien diferenciados independientes el uno del otro. De esta forma lograremos, por ejemplo, que un cambio en la apariencia de la interfaz gráfica no interfiera en el funcionamiento del programa, puesto que los datos seguirán siendo los mismos y la forma de trabajar con ellos tampoco variará.

El **modelo** estará formado por la base de datos SQLite que contiene diversas tablas con la información del estado del *switch*, los parámetros de configuración, lo usuarios registrados en el sistema y los dispositivos conocidos.

La **vista** la conformará la interfaz gráfica, que contará con una ventana principal y una serie de menús y cuadros de dialogo para consultar y modificar los diferentes parámetros y datos.

Por último el **controlador** se encargará de mediar entre la interfaz de usuario y los datos, implementando los diferentes casos de uso vistos en el apartado anterior. Estará formada por dos clases, una encargada de toda la gestión de los usuarios, y otra encargada de lanzar la interfaz gráfica e implementar los casos de uso con los que interactúa el administrador.

Otra característica deseable del diseño es que la aplicación pueda funcionar sin interfaz gráfica. Es decir, la interfaz gráfica, en este caso, se plantea como una he-

rramienta para que el administrador de la red pueda realizar las modificaciones que considere oportunas en el sistema, además de poder consultar quién está conectado al *switch* de forma sencilla. Pero el sistema debe de seguir permitiendo la conexión de nuevos usuarios y realizando la separación lógica en VLAN, manteniendo a los nuevos usuarios conectados en la VLAN restringida hasta que no se hayan autenticado, independientemente de que la interfaz gráfica esté siendo utilizada o no. Es por ello que se hace más necesario aún tener una correcta implementación del patrón modelo-vista-controlador.

3.3.1. Herramientas a utilizar

En esta sección se van a mencionar brevemente las herramientas elegidas para el desarrollo del trabajo, así como el motivo de su elección.

En primer lugar es necesario hablar sobre el lenguaje de programación utilizado para el desarrollo de la aplicación, Python. Se ha elegido este lenguaje de programación debido a su sencillez y a su facilidad de aprendizaje, unido a que dispone de multitud de librerías gráficas para poder elaborar la GUI de la aplicación. Además, cuenta con una librería para trabajar con operaciones SNMP con una extensa documentación, lo cual hace más sencillo su uso, que se llama *pysnmp*. En cuanto a la librería gráfica utilizada para desarrollar la parte gráfica de la aplicación, se ha elegido *PyQt*, ya que es una de las más utilizadas en la actualidad y cuenta con una extensa documentación. Además permite crear de forma relativamente sencilla tablas y conectar señales con sus respectivos *slots*, unido a que cuenta con una herramienta gráfica llamada *QtCreator* para crear la interfaz, que ha supuesto un gran ahorro de tiempo en el aprendizaje del manejo de la librería.

Otro elemento muy importante en la herramienta es la base de datos. Esta base de datos consta de varias tablas, que serán explicadas con detenimiento más adelante, que contienen parámetros de configuración, el estado del *switch* y la relación entre usuarios y contraseñas del sistema. Como la base de datos a utilizar es pequeña y las consultas que se van a realizar sobre ella son bastante sencillas, una buena opción como gestor de base de datos es *SQLite*, ya que es un sistema bastante ligero y *serverless*, ya que no necesita poner en marcha un servidor que haya que configurar previamente y mantener para poder funcionar.

Para finalizar, es necesario mencionar que opciones se han elegido para poner en marcha los diferentes servidores necesarios para el correcto funcionamiento de la herramienta y que han sido mencionados en el inicio del presente capítulo. Para implementar el servidor DNS y DHCP, se ha decidido utilizar *dnsmasq*, ya que es

una herramienta ligera, una de las motivaciones del desarrollo de la aplicación, y porque aúna en él ambos servicios, por lo que no será necesario tener corriendo un proceso por cada servicio. Además será necesario que cualquier consulta DNS realizada desde la VLAN restringida, donde se encuentran los usuarios que aún no han sido autenticados, sea contestada con la dirección IP del servidor web para que así puedan acceder con sus credenciales. En cuanto al servidor web, se ha utilizado Apache HTTP Server.

3.3.2. Diseño de la base de datos

Un elemento fundamental de la aplicación es la base de datos, ya que esta contendrá toda la información necesaria para hacer funcionar la herramienta. Esta información será obtenida del *switch* utilizando el protocolo SNMP o será introducida por el administrador, en los en los que la información a almacenar no esté relacionada con el estado del *switch*. Para ello, es necesario diseñar una base de datos que cuente con varias tablas. Cabe mencionar que en SQLite los tipos de datos booleanos se representan como enteros.

Será necesario crear un total de 4 tablas, para así poder guardar y obtener los datos de manera más sencilla, sin mezclar en una misma tabla elementos que no tengan que ver entre sí.

La primera tabla contiene la información acerca del estado actual de los puertos del *switch* y será denominada **switch**. Por ello, deberá contar con un campo entero, que además será la clave, que identifique el número de puerto. Además, deberá contar también con un campo que guarde el estado en el que se encuentre el *switch*, otro con la dirección MAC del dispositivo conectado a ese puerto, si es que lo hubiera, y un campo con el *login* del usuario conectado a ese puerto. Por último es necesario incluir un campo donde guardar el identificador de VLAN del puerto en cuestión y un campo auxiliar que, en función de su valor, indique si el estado del puerto está siendo manipulado por la aplicación. A continuación se muestra en detalle cada uno de los campos de la tabla.

- **puerto:** campo entero que identifica el número de puerto. Es la clave primaria.
- **estado:** campo de tipo entero que refleja el estado del puerto del *switch*. Si su valor es 2 significa que el puerto está desconectado, mientras que si su valor es 1 el puerto está conectado. Se ha utilizado esta notación porque son los valores que utiliza SNMP para guardar el estado de los puertos.

- **login:** campo de tipo texto que almacena el usuario registrado conectado a un determinado puerto. Si el puerto estuviera desconectado o no se hubiera identificado aún el usuario, su valor será *NULL*.
- **mac:** campo de tipo texto que sirve para guardar la dirección MAC del dispositivo conectado al puerto. Si no hubiera ningún dispositivo conectado al puerto su valor será *NULL*.
- **puerto_manipulado:** campo de tipo entero que se utiliza para reflejar que el cambio en el estado del *switch* que ha provocado la llegada de una trampa, ya sea *linkUp* o *linkDown*, no ha sido espontáneo, sino que ha sido debido a una acción llevada a cabo por la aplicación, por lo que debe de ser ignorado. Cuando este campo tiene su valor a 0, significa que no ha habido manipulación alguna en el *switch* por parte de la aplicación. Si por el contrario su valor es 1, significa que la trampa debe ser ignorada.
- **vlan:** campo de tipo entero que contiene el identificador de VLAN asociado al puerto.

Los campos de esta tabla se irán actualizando según vayan sucediendo los eventos definidos en los casos de uso. Por ejemplo, cuando se reciba una trampa de tipo *linkUp*, se cambiará el estado de la entrada correspondiente de la tabla a 1 y se añadirá la dirección MAC del dispositivo, mientras que si el evento ocurrido es la autenticación de un usuario, se añadirá su *login* y se cambiará el contenido del campo *vlan*.

La segunda tabla será la que contenga a los usuarios registrados en el sistema. Deberá contar con dos campos, el *login* del usuario y su contraseña. Esta tabla será utilizada para cotejar los datos introducidos por un usuario en el proceso de autenticación. De esta forma, una vez el usuario ha introducido sus credenciales, se busca en esta tabla un par usuario-contraseña idéntico al introducido por el cliente. El nombre de la tabla es **usuarios**.

- **login:** campo de texto que contiene el nombre del usuario. Es la clave primaria.
- **password:** campo de texto que contiene la contraseña del usuario.

La tercera tabla contiene la dirección MAC de los dispositivos conocidos por el sistema, además del nombre del equipo. Esta tabla será consultada cuando se produzca la recepción de una trampa SNMP indicando que se ha conectado un

dispositivo a un puerto del *switch*. Una vez obtenida la MAC del dispositivo se busca en esta tabla y si se encuentra, el dispositivo conectado a ese *switch* es cambiado inmediatamente a la VLAN de dispositivos conocidos. Además, se añade en la tabla `switch` el nombre del dispositivo en el campo *login* de la entrada correspondiente y se cambia el valor del campo *vlan*. El nombre de esta tabla es **dispositivos_conocidos**.

- **mac:** campo de tipo texto que contiene la dirección MAC del dispositivo. Es la clave primaria.
- **nombre:** campo de tipo texto que contiene el nombre de la máquina asociado a la dirección MAC.

La cuarta y última tabla es la que contiene los elementos de configuración de la aplicación. Principalmente contiene elementos relacionados con el acceso al *switch* mediante SNMP, aunque también incluye aspectos como el identificador de VLAN de las tres redes que conforman la red y el identificador del proceso en el cual está corriendo la interfaz gráfica. Esta tabla contará con una única entrada, y no tendrá ninguna relación con el resto de tablas. El nombre de la tabla es **configuracion** y solo podrá ser modificada en el cuadro de diálogo destinado a que el administrador cambia los parámetros de configuración, a excepción del campo que contiene el pid, que será modificado siempre que se abra y cierre la interfaz gráfica. A continuación se especifican los campos que la conforman.

- **ip:** Campo de tipo texto que contiene la dirección IP del *switch*.
- **num_puertos:** campo de tipo entero que contiene el número de puertos que tiene el *switch*.
- **pid_proceso:** campo de tipo entero que contiene el identificador de proceso asociado a la interfaz gráfica. Este campo servirá para poder enviar la señal UNIX a la interfaz gráfica que indica que se ha producido un cambio en el estado del *switch*.
- **vlan_restringida:** campo de tipo entero que contiene el identificador de la VLAN en la que se encuentran los usuarios desconocidos.
- **vlan_conocidos:** campo de tipo entero que contiene el identificador de la VLAN en la que se encuentran los dispositivos conocidos por el sistema.
- **vlan_registrados:** campo de tipo entero que contiene el identificador de la VLAN en la que se encuentran los usuarios identificados.

- **comunidad_lectura:** campo de tipo texto que contiene la comunidad SNMP para acceder al switch en modo lectura.
- **comunidad_escritura:** campo de tipo texto que contiene la comunidad SNMP para acceder al switch en modo escritura.

3.3.3. Diseño de la interfaz gráfica

A lo largo de la descripción de los casos de uso, se han ido mencionando poco a poco algunos de los componentes que deben componer la interfaz gráfica. Contará con una ventana principal en la que se mostrará el estado de los puertos del *switch* de forma gráfica y una serie de cuadros de dialogo o subventanas con los que el administrador podrá interactuar y que han sido mencionados con anterioridad.

Como se ha elegido PyQt como herramienta para el desarrollo de esta parte de la aplicación, cada ventana que conforma la GUI constituirá una clase de la aplicación.

Clase MainWindow

La ventana principal será el elemento fundamental de la aplicación. Contendrá un menú con las opciones que tiene el administrador para consultar y modificar datos, además de la representación gráfica del estado del *switch*.

La mayor parte de esta ventana estará ocupada por el *switch*. Cada puerto del *switch* estará numerado y se unirá a un icono de un ordenador, que representa el dispositivo conectado. Debajo de cada icono de ordenador se situará un cuadro de texto que contendrá la información más relevante acerca del estado del puerto.

Las líneas que simulan la conexión entre el *switch* y el dispositivo conectado podrán tomar tres colores diferentes. Si en el puerto en cuestión no hay ningún *host* conectado, el color de la línea será gris. Si hay algún *host* conectado, pero este se encuentra en la VLAN de usuarios desconocidos, el color del cable será amarillo, mientras que si el usuario conectado al puerto se ha identificado, o por el contrario el dispositivo conectado se trata de un *host* conocido, el color del cable será verde. Con este diseño se pretende que de un simple vistazo sea posible conocer los puertos que están siendo utilizados y si los usuarios conectados están autenticados o no.

Para complementar la información que arroja el color del cable que sale de

cada puerto del *switch*, se ha incluido un cuadro de texto debajo de cada ordenador, ampliando la información ofrecida por el código de colores de los cables. Si el puerto está desconectado, en el cuadro de texto solamente pondrá 'Desconectado'. Si ,por el contrario, hay algún dispositivo conectado al puerto, se mostrará el usuario conectado y debajo la dirección MAC del dispositivo. En caso de que el usuario conectado no se haya autenticado, se mostrará que el usuario es desconocido.

Como se ha comentado anteriormente la ventana principal debe contar con un menú. Las opciones con las que cuenta el menú son las siguientes:

- **Ver puertos:** opción que sirve para abrir una subventana que contiene una tabla con el estado detallado de los puertos del *switch*.
- **Inicializar BD:** opción que sirve para actualizar o inicializar la base de datos obteniendo directamente los datos del switch mediante una serie de operaciones SNMP.
- **Gestionar switch:** opción que sirve para abrir un cuadro de dialogo y modificar los datos contenidos en la tabla `configuracion` de la base de datos.
- **Actualiza interfaz:** opción que sirve para refrescar la ventana principal.
- **Cerrar:** opción que cierra la interfaz gráfica de la aplicación.

Con todo lo expuesto, en la figura 3.2 se muestra un primer boceto del diseño deseado para la ventana principal del programa.

Clase Gestion

Esta clase proporciona al administrador una herramienta para poder cambiar los ajustes de configuración del *switch*. Es importante tener en cuenta que la información que se modifica a partir de este cuadro de dialogo es la que consta en la base de datos, por lo que no se cambian los ajustes en el *switch*, sino que se cambian los parámetros que figuran en la base de datos y que son con los que trabaja la herramienta.

En cuanto a la apariencia, esta clase no es más que un cuadro de dialogo que muestra los elementos de la base de datos que pueden ser modificados, además de mostrar su valor actual. Deberá contar también con un botón que sirva para guardar los cambios y otro botón que sirva para cancelar los cambios que se hayan realizado y volver a mostrar los datos que hay guardados en la base de datos.

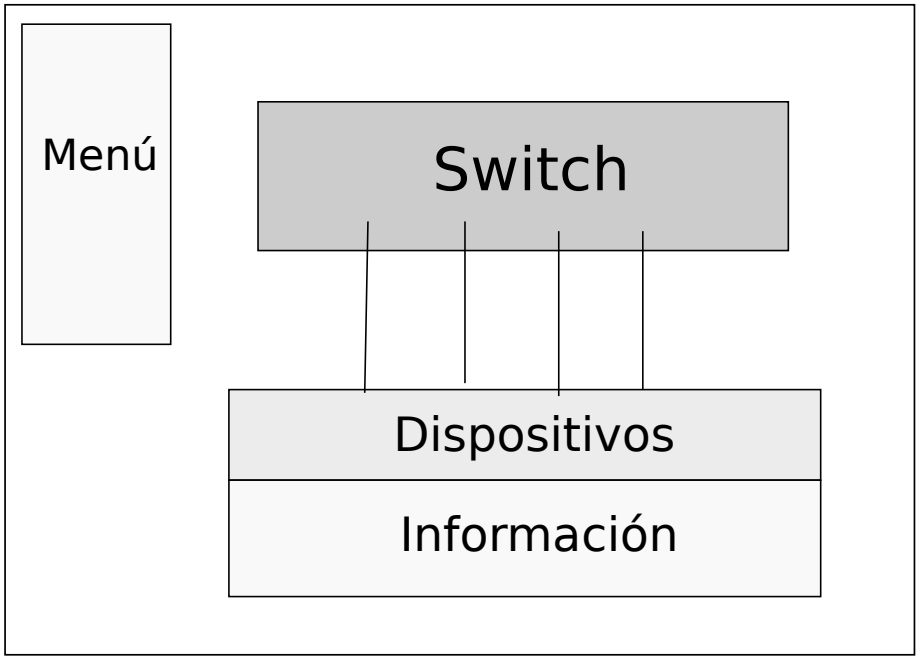


Figura 3.2: Boceto de la interfaz principal.

Los datos que pueden ser modificados desde esta clase son aquellos que figuran en la tabla `configuracion` de la base de datos, a excepción del campo *pid*, que no es un parámetro configurable por el administrador.

En cuanto al contenido del cuadro de diálogo, no es más que una línea que contenga el nombre del parámetro a modificar y un cuadro de introducción de texto por cada elemento, además de los dos botones comentados anteriormente. Los cuadros de introducción de texto estarán inicializados con el valor actual del campo. En la figura 3.3 se puede ver un ejemplo de como debería ser esta ventana.

Por último, faltaría una clase para contener la subventana en la que se muestra de forma detallada el estado de los puertos. Como esta información va a ser mostrada en forma de tabla, no es necesario crear una clase para contener la tabla, así que se ha diseñado como un método dentro de la clase *Aplicacion*, que comentaremos más adelante. Esto es posible ya que *PyQt* cuenta con un objeto para dibujar tablas sin la necesidad de crear una nueva clase para mostrar la subventana que contiene la tabla.

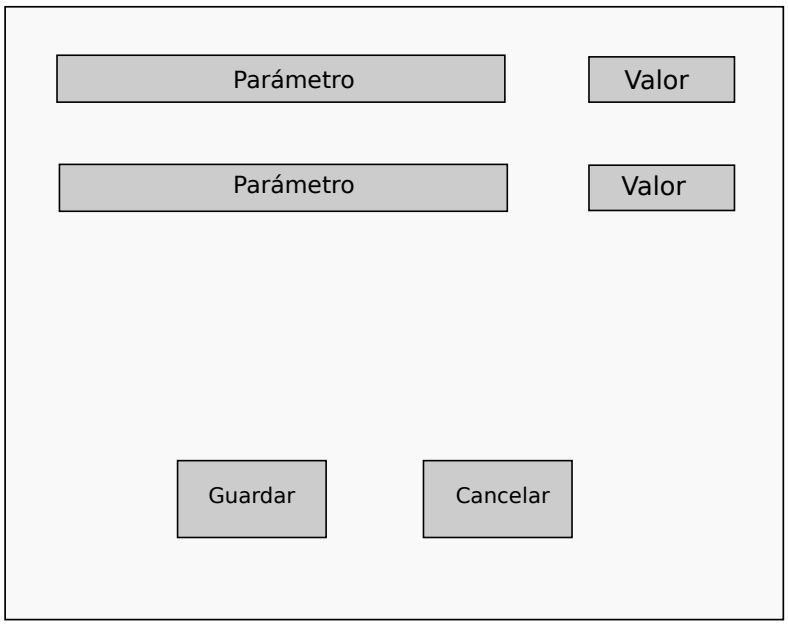


Figura 3.3: Boceto de la ventana para modificar la base de datos.

3.3.4. Diseño del controlador

Una vez definido el diseño de los dos primeros componentes del patrón *MVC*, es necesario continuar con el controlador. Como se ha mencionado anteriormente, esta parte es la encargada de mediar entre la interfaz y la base de datos, realizando la mayor parte de las operaciones. El controlador estará formado por dos clases. La primera será la encargada de ejecutar la interfaz gráfica y realizar las operaciones solicitadas por el administrador. La segunda clase es la encargada de gestionar las notificaciones enviadas por el *switch*, así como de modificar el estado de la tabla *switch* de la base de datos.

Clase GestionVlanSNMP

Esta clase es la encargada de gestionar la conexión y desconexión de un puerto, además de la autenticación de un usuario. Es llamada por el demonio *snmptrapd* cada vez que recibe una trampa y por el servidor *apache* cuando un usuario se autentica correctamente.

Como se ha mencionado anteriormente, el sistema de control de acceso debe seguir funcionando aunque la interfaz gráfica no se esté ejecutando. Es por ello que

esta clase debe estar en un *script* aparte del resto de la aplicación, para así poder ser ejecutado por los diferentes demonios.

La clase contiene todos los métodos necesarios para realizar el control de acceso a red y mantener a cada usuario en la VLAN que le corresponda. Además, sus variables de clase son aquellos los que figuran de la tabla *configuracion* de la base de datos, puesto que es la única clase que interactúa directamente con el *switch* y necesita saber su dirección IP, los identificadores de puerto de las diferentes redes y las comunidades SNMP de acceso al *switch*.

Clase Aplicacion

La segunda clase que conforma el controlador es la encargada de gestionar las peticiones realizadas por el administrador de red, además de lanzar la interfaz gráfica. Su función es crear una instancia de la ventana principal de la aplicación y capturar los eventos producidos para su posterior tratamiento. No debería haber dos instancias de la aplicación ejecutándose a la vez, por lo que antes de lanzar la interfaz gráfica es necesario comprobar que no existe ya un proceso ejecutándose. Por ello es necesario comprobar que el campo de la base de datos que contiene el PID de la aplicación está vacío y por supuesto, cuando se cierre la aplicación deberá dejarse ese campo a *null*, ya que si no, al volver a ejecutar la aplicación no podrá arrancar.

Otro aspecto deseable es introducir una opción a la hora de ejecutar la aplicación gráfica, que permita forzar su arranque, independientemente de que haya ya un proceso corriendo o no.

En la figura 3.4 se puede ver un diagrama de clases simplificado de la aplicación.

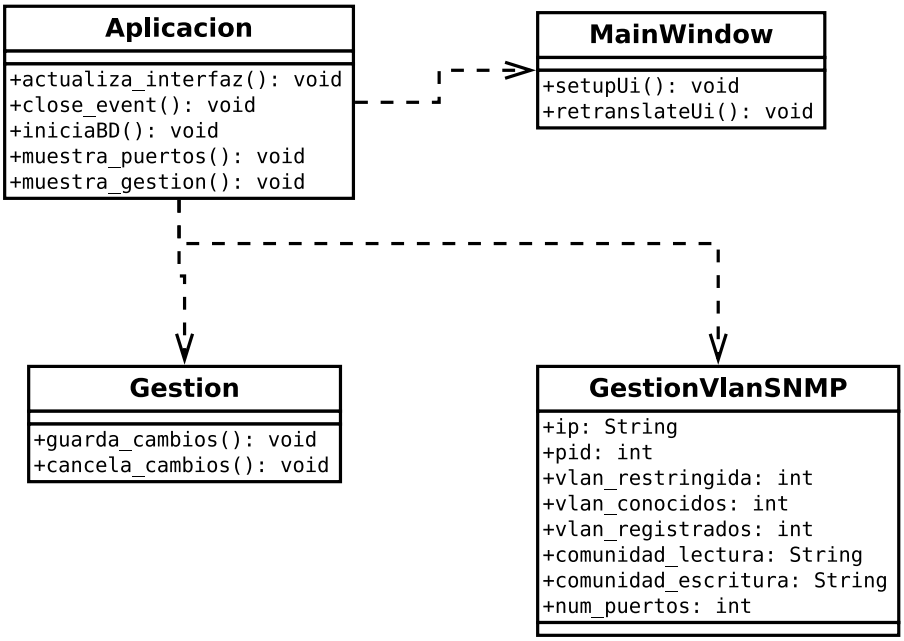


Figura 3.4: Diagrama de clases simplificado de la aplicación.

Capítulo 4

Desarrollo de la aplicación

Una vez finalizada la fase de diseño de la aplicación, el siguiente paso es proceder a su desarrollo. Para ello, además de la implementación de las clases y los casos de uso expuestos en el anterior capítulo, es necesaria la puesta en marcha y configuración de todos los servidores necesarios para que la herramienta funcione correctamente. Este tema se abordará en el quinto capítulo de la presente memoria.

En este capítulo se va a mostrar el desarrollo *software* utilizando las herramientas mencionadas en capítulos anteriores.

4.1. Implementación de los casos de uso

4.1.1. Implementación de caso de uso LinkUp

Este caso de uso es provocado por la llegada de un trampa del tipo *linkUp* al servidor SNMP. Cuando el servidor SNMP recibe este tipo de trampa, llama al script *gestionVlanSNMP.py*, que contiene la clase que da nombre al fichero, con los parámetros correspondientes. Estos parámetros serán el estado del puerto, que en este caso es *up*, y el puerto que ha originado la trampa.

Una vez se comprueba en la función main del fichero mencionado anteriormente que se trata de un levantamiento de puerto, se realiza una llamada al método *link_up()*, pasándole como argumento el puerto. Dentro de esta función lo primero que se hace es llamar al método *Obtiene_MAC()*, que recibe como parámetro el

```
def obtieneMAC(self,puerto):
 #Hace la consulta SNMP para obtener la relación de puerto-mac que hay en el switch
 respuesta,valor = self.snmpwalk("1.3.6.1.2.1.17.7.1.2.2.1.2")
 lista = []
   for i in range(0,len(valor)):
     #Si el resultado que se está evaluando es el del puerto que queremos obtener la MAC entra
     if(valor[i]==int(puerto)):

       #Separa la parte de la MAC de la parte del OID
       auxiliar  = str(respuesta[i]).split("1.3.6.1.2.1.17.7.1.2.2.1.2.")
       decimal = auxiliar[1].split(".")
       #Empieza en 1 porque el primer elemento corresponde a la vlan del puerto
       for i in range (1,len(decimal)):
       #Guarda en una lista cada parte de la dirección MAC
       #La función hex() devuelve el resultado de la forma "0x"
       #Con la función replace lo eliminamos
       if int(decimal[i]) > 15:

         lista.append(hex(int(decimal[i])).replace("0x",""))
         #Si el valor es menor de 15 (F), coloca un 0 delante
         #para que cada octeto cuente con 2 dígitos
       else:

         lista.append("0" + hex(int(decimal[i])).replace("0x",""))
         return ":".join(lista)

 return None
```

Figura 4.1: Método ObtieneMAC

puerto también.

Para obtener la dirección MAC del puerto es necesario realizar una operación *walk* sobre el OID *dot1qTpFdbPort* [IET06], que contiene la información de la dirección física conectada a cada puerto del *switch*. En la tabla 4.1 se puede observar la forma de obtener la dirección MAC. Como es posible que la dirección MAC del dispositivo conectado tarde cierto tiempo en ser registrada, es necesario realizar varios reintentos hasta obtener un parámetro distinto de *None*. Una vez obtenida la dirección MAC se comprueba si la trampa ha sido provocado por la aplicación, lo cual puede ocurrir en algunas ocasiones, como se verá en la sección 2.2 de este capítulo. Para comprobar esto no hace falta más que consultar el estado del campo *puerto_manipulado* de la tabla `switch` de la base de datos. Si resulta que la trampa ha sido provocada, el método termina.

En el caso de que la trampa se haya producido por la conexión de un dispositivo al puerto, se comprueba si la dirección del dispositivo figura en la base de datos y, si es así, se le cambia de VLAN, mediante el método *cambia_vlan()*, que será explicado en la sección 2.2 de este capítulo.

Por último se cambia el estado del puerto en la base de datos y se añaden el resto de datos recopilados (dirección MAC, *login* en el caso de ser un dispositivo conocido y dirección MAC) y se lanza la señal para actualizar la interfaz, como se puede ver en la figura 4.2. El envío de esta señal a la interfaz gráfica es necesario

```
try:
    os.kill(self.pid, signal.SIGUSR1)
except:
    print "El proceso ya no existe"
```

Figura 4.2: Envío de señal UNIX SIGUSR1.

para que se pueda refrescar la ventana principal de la aplicación cada vez que ocurre un cambio. Cabe mencionar también que esta señal no será enviada si el campo *puerto_manipulado* tiene valor 1.

4.1.2. Implementación del caso de uso LinkDown

Este caso de uso es muy similar al anterior, ya que su objetivo es deshacer los cambios realizados y dejar la base de datos y el puerto del *switch* tal y como estaba antes de la conexión del dispositivo que se ha desconectado.

La sucesión de eventos que desencadenan este caso de uso es la misma. El servidor SNMP recibe una trampa, que en este caso indica la desconexión de un puerto, y llama al *script* python que contiene la clase *gestionVlanSNMP*. En la función main de este *script* se comprueban los argumentos y se llama al método *link_down()*, que se encarga de gestionar la desconexión de un puerto.

La primera tarea que debe realizar este método es la comprobación del estado del campo *puerto_manipulado* de la tabla `switch` de la base de datos, para comprobar si realmente se ha desconectado un dispositivo. Si nada extraño ha ocurrido, el siguiente paso será cambiar la entrada de la tabla `switch` correspondiente al puerto que ha provocado la trampa. Para ello será necesario cambiar el *estado* a 2, y poner a null los campos *login* y *mac*. Se ha elegido este valor del campo *estado* ya que es el mismo que utiliza el protocolo SNMP para indicar que un puerto está desconectado. En la figura 4.3 se puede ver como se realiza este proceso, siendo resultado[0] el valor del campo que indica si el puerto ha sido manipulado por el programa.

Para finalizar, además de enviar la señal UNIX para notificar el cambio en el estado de uno de los puertos, es necesario comprobar si hay que cambiar al puerto de VLAN, puesto que este proceso debe dejarlo en la VLAN restringida. Para ello, se debe consultar el campo *vlan* de la base de datos para ver cual es su identificador. Si el puerto ya se encuentra en la VLAN restringida, no se hace nada y finaliza

```
if resultado[0] == 1:
            con.close()
            #Significará que la desconexión del puerto la hemos provocado nosotros
            return #Lo deja todo como está
        else:
            try:

                sql_update = "UPDATE switch SET estado = 2, mac = null, vlan ="
                +str(self.vlan_restringida) +", login=null WHERE puerto = " + str(puerto)
                cursorObj.execute(sql_update)
                con.commit()

            except:

                print "Ha ocurrido un error al actualizar el estado del puerto %s" %puerto
                return
```

Figura 4.3: Cambio en la base de datos por desconexión.

```
sql_consulta = "SELECT vlan FROM switch where puerto = " + puerto
cursorObj.execute(sql_consulta)
vlan = cursorObj.fetchone()
if vlan[0] != self.vlan_restringida:
     #Cambia el puerto de VLAN
      self.cambia_vlan(puerto,vlan[0],self.vlan_restringida)
con.close()
```

Figura 4.4: Obtención y cambio de VLAN.

el método. De no ser así, es necesario cambiar al puerto de VLAN, y cambiar el estado del campo en cuestión de la tabla que contiene la información sobre los el estado actual de los puertos del *switch*. En la figura 4.4 se muestra como se realiza este proceso. En este caso, es la función *cambia_vlan()* la encargada de modificar la base de datos.

4.1.3. Implementación de caso de uso CambiaValores

Como se puede observar en la tabla 3.5, este caso de uso es iniciado por el administrador, a diferencia de los dos anteriores que eran consecuencia de un cambio en el agente SNMP. Además, también es el primero de los casos de uso analizados en el cual interviene la interfaz gráfica, algo que debe ocurrir si se encuentra entre los agentes implicados el administrador.

El caso de uso comienza cuando el administrador pulsa el botón 'Gestionar *switch*' del menú de la ventana principal de la aplicación. Una vez realizada esta acción se abre un cuadro de diálogo con el cual se pueda interactuar e introducir los cambios.

La clase *Aplicacion* creará una instancia de la clase *Gestion* y mostrará la ventana. Esta clase crea un formulario y añade una serie de *labels* y *QLineEdit*,

```
        Form.setObjectName(_fromUtf8("Form"))
        Form.setGeometry(0, 0, 400, 360)
        Form.setStyleSheet(_fromUtf8("background-color:
         rgb(231, 231, 231);"))
        Form.setMaximumSize(QtCore.QSize(400, 360))
        Form.setWindowTitle(_translate("Form", "Gestión de los
        parametros del switch", None))
```

Figura 4.5: Creación del cuadro de dialogo de la clase Gestion.

uno por cada parámetro a modificar. En PyQt, una *label* es un cuadro de texto, que no admite modificación por parte del usuario que está utilizando la aplicación, mientras que un *QLineEdit* es la herramienta con la que cuenta la librería gráfica para la introducción de datos por parte del usuario. En la figura 4.5 se muestra como crear el formulario, estableciendo el tamaño de la ventana y el color de fondo, además de su tamaño máximo y el título de la ventana.

Una vez creada la ventana, es necesario crear los elementos que la van a conformar. Como se mencionó unas lineas antes, se basará en una serie de etiquetas para identificar el elemento a modificar, junto con un elemento de introducción de texto y los botones para guardar y cancelar los cambios. Los *QLineEdit*, deberán tener por defecto el valor actual de los parámetros. Es por ello que será necesario realizar una conexión a la base de datos para obtenerlos. En la figura 4.6 aparece reflejada la creación de estos dos elementos mientras que en la figura 4.7 se muestra la forma dar valor a los cuadros de texto.

Para completar esta parte de la interfaz, faltaría crear los botones de guardar y cancelar. Esto se puede realizar creando dos objetos *QtGui.QPushButton*. Con todos estos componentes, el resultado del cuadro de diálogo para modificar los datos de gestión de la aplicación sería el que se puede observar en la figura 4.8.

Hasta el momento se ha explicado la creación de la ventana de configuración, pero no se ha hecho referencia a ningún método que aborde la pulsación de uno de los dos botones que aparecen en la ventana. Esto es debido a que están asociados al evento de pulsación de cada uno de los botones y se encuentran dentro de la clase *Aplicacion*, como ya se mencionó en el apartado de diseño.

Lo primero que se debe hacer es conectar la señal de pulsación de cada uno de los botones con su *slot* correspondiente, que en este caso son los métodos *cancela_cambios()* y *guarda_cambios()*. En la figura 4.9 se muestra cómo se debe proceder para conectar estos dos elementos de la interfaz.

```
        #Crea la etiqueta y  fija el tamano y posición
        self.label = QtGui.QLabel(Form)
        self.label.setGeometry(QtCore.QRect(20, 40, 231, 18))
        self.label.setObjectName(_fromUtf8("label"))
        #Crea el campo de texto y fija el tamano y posición
        self.ip = QtGui.QLineEdit(Form)
        self.ip.setGeometry(QtCore.QRect(250, 40, 113, 21))
        self.ip.setObjectName(_fromUtf8("ip"))
        self.ip.setStyleSheet(_fromUtf8("background-color:
         rgb(255, 255, 255);"))
         #Rellena la etiqueta
        self.label.setText(_translate("Form", "<html><head/>
         <body><p><span style=\" font-weight:600;\">Dirección
         IP:</span></p></body></html>", None))
```

Figura 4.6: Creación de etiqueta y QLineEdit para la dirección IP.

```
#Realiza la conexión a la base de datos
con = sqlite3.connect("/home/dario/TFG/basedatos.db")
cursorObj = con.cursor()
#Realiza la consulta para obtener el estado de los puertos
sql_consulta = "SELECT * FROM configuracion"
cursorObj.execute(sql_consulta)
row = cursorObj.fetchone()
self.ip.setText(_translate("Form", str(row[0]), None))
self.num_puertos.setText(_translate("Form", str(row[1]), None))
self.vlan_restringida.setText(_translate("Form", str(row[3]), None))
self.vlan_conocidos.setText(_translate("Form", str(row[5]), None))
self.vlan_registrados.setText(_translate("Form", str(row[4]), None))
self.comunidad_lectura.setText(_translate("Form", str(row[6]), None))
self.comunidad_escritura.setText(_translate("Form", str(row[7]), None))

con.close()
```

Figura 4.7: Obtención y muestra de los datos actuales de configuración.

Figura 4.8: Ventana de configuración.

```
QtCore.QObject.connect(self.ui.guardar, QtCore.SIGNAL(QtCore.QString.fromUtf8("clicked()")),
self.guarda_cambios)
QtCore.QObject.connect(self.ui.cancelar, QtCore.SIGNAL(QtCore.QString.fromUtf8("clicked()")),
self.cancela_cambios)
```

Figura 4.9: Conexión de señal con su *slot*.

El método *guarda_cambios()* se encarga de recoger el contenido de los campos de texto y realiza una consulta de actualización de la tabla `configuracion` de la base de datos. En caso de que el número de puertos sea uno de los parámetros modificados, se deberá cambiar también el tamaño de la tabla `switch`, añadiendo o quitando registros, en función de si el dato nuevo es mayor o menor al que había guardado. Para obtener los datos de los campos de texto se aplica el método *text()*, sobre el objeto que define el QLineEdit. Al finalizar este método se muestra una ventana emergente, creada con la clase *QMessageBox*, para indicar que los cambios han sido guardados con éxito y se cierra la ventana.

El método *cancela_cambios()* no es más que volver a escribir los datos que constan en la base de datos sobre los cuadros de texto.

4.1.4. Implementación del caso de uso ActualizaBD

A pesar de que este método se origina debido a la pulsación de la opción 'Inicializa DB' del menú de la ventana principal de la aplicación, no abre ninguna ventana

```
        for i in range(0,int(self.num_puertos)):
            mac= self.obtieneMAC(i+1)
            if mac != base[i][3]:
                if mac:

                    cambio = 1
                    sql_update= "UPDATE switch SET mac ='" + mac + "' WHERE puerto=" + str(i+1)
                    cursorObj.execute(sql_update)
                    con.commit()
                else:
                    cambio = 1
                    sql_update= "UPDATE switch SET mac = null WHERE puerto=" + str(i+1)
                    cursorObj.execute(sql_update)
                    con.commit()
```

Figura 4.10: Actualización del campo *mac*.

nueva, y las operaciones realizadas se llevan a cabo en la clase *gestionVlanSNMP*, no en la clase *Aplicacion*. Esto se debe a que este caso de uso interactúa directamente con el *switch*, y ahí es donde se encuentran los métodos necesarios para ello.

El evento de pulsación de la opción del menú principal está conectada con un método de la clase *Aplicacion* llamado *InicializaBD*. Este método lo que hace es crear una instancia de la clase *gestionVlanSNMP* y se invoca al método *importa_datos* para actualizar el estado de la base de datos, si es que hiciera falta. Este método devuelve 0 si la base de datos estaba actualizada, mientras que si su valor es distinto significará que la información de la base de datos no reflejaba correctamente el estado de los puertos del *switch*.

En cuanto al método *importa_datos*, que se encuentra en el script *gestionVlanSNMP.py* y obtiene todos los datos de la tabla `switch`, a excepción del campo que contiene el identificador de usuario, que no se puede obtener mediante SNMP.

Para obtener el estado de cada puerto es necesario realizar una operación *walk* sobre el nodo ifOperStatus. La respuesta obtenida será una matriz que contiene el número de puerto y su estado, además de otros parámetros. El estado de cada puerto se compara con el que figura en la base de datos y, si difiere, se cambia.

Para obtener la dirección MAC se opera de la misma forma, pero en este caso, como ya se dispone de un método de la clase para obtener la dirección MAC de un puerto, basta con invocarlo. Si el resultado devuelto por esta función difiere del guardado en la base de datos, se cambia por el obtenido. En la figura 4.10 se puede observar como se realiza la comprobación, siendo similar a la del resto de campos de la tabla.

Para finalizar la actualización de la base de datos, es necesario obtener el iden-

tificador VLAN de cada puerto. Para ello es necesario realizar un `snmpwalk` sobre el nodo *dot1qPvid* y proceder de la misma manera que en casos anteriores.

4.1.5. Implementación del caso de uso MuestraDetallePuertos

Debido a que la ventana principal no muestra todos los detalles acerca del estado de los puertos del *switch*, es necesario contar con un mecanismo para que el administrador pueda acceder a estos datos desde la interfaz principal. Esta información será presentada en forma de tabla cuando la opción del menú 'Ver puertos' sea pulsada, por lo que habrá que conectar esta señal con su *slot* correspondiente, que en este caso será el método *muestra_puertos()* de la clase *Aplicacion*.

Para llevar a cabo la creación de esta nueva ventana es necesario crear un elemento *QTableWidget* con 5 columnas y tantas filas como puertos tenga el *switch* que opera en la red. Una vez proporcionado un nombre a la nueva ventana y añadido el nombre de las columnas de la tabla recién creada, es momento de obtener la información que se va a mostrar. Para ello se realiza una consulta a la base de datos que extraiga todos los datos de la tabla `switch` y se añaden a la tabla. Con el fin de que la tabla sea más legible, será necesario realizar una serie de cambios a estos datos obtenidos, como por ejemplo mostrar conectado o desconectado en lugar del valor entero que se guarda en la tabla, en el campo *estado*.

En la figura 4.11 se como realizar este proceso, teniendo en cuenta que las variables *row[4], row[5] y row[6]*, contienen el PVID de cada VLAN, mientras que en la figura 4.12 se muestra el resultado final obtenido.

4.1.6. Implementación del caso de uso AccesoSistema

Este caso de uso se produce cuando un usuario conectado a la red introduce sus credenciales de forma correcta. Para que esto suceda es necesario haber accedido a la web de acceso y haber cotejado los datos introducidos con los que figuran en la base de datos. Todo este proceso será explicado en el siguiente capítulo, mientras que en esta sección se va a explicar lo que sucede una vez el *script* PHP *(Hypertext Preprocessor)* notifica a la aplicación que un usuario conectado al sistema se ha autenticado con éxito.

Una vez realizada la comprobación de que las credenciales son correctas, se

```
sql_consulta = "SELECT puerto, estado, login, mac, vlan FROM switch"
cursorObj.execute(sql_consulta)
rows = cursorObj.fetchall()
        i = 0
for fila in rows:
            self.table.setItem(i, 0, QtGui.QTableWidgetItem(str(fila[0])))
            if fila[1] == 2:
                self.table.setItem(i, 1, QtGui.QTableWidgetItem("Desconectado"))
            else:
                self.table.setItem(i, 1, QtGui.QTableWidgetItem("Conectado"))
                self.table.setItem(i, 2, QtGui.QTableWidgetItem(str(fila[2])))
                self.table.setItem(i, 3, QtGui.QTableWidgetItem(str(fila[3])))
            if fila[4] == row[3]:
                item = "Restringida(" + str(row[3]) + ")"
                self.table.setItem(i, 4, QtGui.QTableWidgetItem(item))
            elif fila[4] == row[5]:
                item = "Conocidos(" + str(row[5]) + ")"
                self.table.setItem(i, 4, QtGui.QTableWidgetItem(item))
            elif fila[4] == row[4]:
                item = "Registrados(" + str(row[4]) + ")"
                self.table.setItem(i, 4, QtGui.QTableWidgetItem(item))
            else:
                 self.table.setItem(i, 4, QtGui.QTableWidgetItem(str(fila[4])))
            i = i + 1
        con.close()
        self.table.show()
```

Figura 4.11: Obtención y relleno de la tabla que muestra los puertos.

Estado del Switch

Puerto	Estado	Usuario	Direccion MAC	VLAN
1	Conectado	router	28:6c:07:41:d8:3b	10
2	Conectado	admin	f8:a9:63:de:39:da	10
3	Desconectado			Restringida(300)
4	Desconectado			Restringida(300)
5	Conectado	dario	00:26:9e:d0:51:deS	Conocidos(200)
6	Conectado	Desconocido	b8:27:eb:7a:ca:c9S	Restringida(300)
7	Desconectado			Restringida(300)
8	Desconectado			Restringida(300)

Figura 4.12: Resultado final de la tabla que muestra el estado del *switch*.

llama al *sript* Python *gestionVlanSNMP.py* con tres parámetros, el primero será la cadena de texto 'login' para que en el main de dicho *script* se pueda saber qué ha motivado la ejecución del *script*; el segundo será el puerto al que está conectado el usuario y el tercero el *login* con el que está registrado. Dentro del *script* se invocará el método *usuario_autenticado()*, que recibe como parámetros el puerto y el *login*. Este método se encargará de modificar la base de datos y cambiar al puerto de la VLAN restringida a la VLAN de usuarios autenticados. Por último se envía la señal UNIX de la misma forma que en los casos de uso LinkUP y LinkDown.

```
if row[1] == 1:
     #Si se encuentra en la VLAN 300 pone el cable de color amarillo
     if row[5] == 300:
          self.ui.line_1.setStyleSheet(_fromUtf8("background-color: rgb(255, 255, 0);\n"
               "color: rgb(255, 255, 0);"))
          texto = """<html><head/><body><p><span style=\" font-weight:600;\">Usuario: </span>
               Desconocido</p><p>""" + str(row[3]) + "</p></body></html>"
          #Muestra la información
          self.ui.datos_1.setText(_translate("MainWindow", texto, None))
     else:
          #Si pertenece a cualquier otra VLAN, cambia el color a amarillo
          self.ui.line_1.setStyleSheet(_fromUtf8("background-color: rgb(0, 255, 0);\n"
               "color: rgb(0, 255, 0);"))
          #Si no hay usuario registrado muestra el usuario como Desconocido
          if row[2] == None:
              texto = """<html><head/><body><p><span style=\" font-weight:600;\">Usuario: </span>
                Desconocido</p><p>""" + str(row[3]) + "</p></body></html>"

              #Muestra la información
              self.ui.datos_1.setText(_translate("MainWindow", texto, None))
          else:
              texto ="""<html><head/><body><p><span style=\" font-weight:600;\">Usuario:
                 </span>""" + str(row[2]) + "</p><p>" + str(row[3]) + "</p></body></html>"
              self.ui.datos_1.setText(_translate("MainWindow", texto, None))

#Si el puerto está desconectado entra
else:
     #Pinta la línea de gris
     self.ui.line_1.setStyleSheet(_fromUtf8("background-color: rgb(173, 173, 173);\n"
         "color: rgb(173, 173, 173);"))
     #Muestra la información
     self.ui.datos_1.setText(_translate("MainWindow", """<html><head/><body><p>Desconectado
         </p><p><br/></p></body></html>""", None))
```

Figura 4.13: Código para actualizar un puerto de la interfaz.

4.1.7. Implementación del caso de uso ActualizaInterfaz

Este caso de uso puede ser provocado por dos motivos: ha ocurrido un cambio en el estado de uno de los puertos del *switch* o el administrador ha elegido la opción 'Actualizar interfaz' del menú de la ventana principal.

Cuando ocurre uno de estos dos eventos, se invoca el método *actualiza_interfaz()* de la clase *Aplicacion*, cuyo cometido es obtener de nuevo los datos del estado del switch para repintar las líneas que simulan los cables de conexión y rellenar de nuevo los cuadros de texto que contienen la información.

Una vez realizada la consulta para obtener los datos del *switch*, es necesario ir puerto a puerto completando los datos que se muestran en la ventana principal de la aplicación. De esta forma, y como se muestra en la figura 4.13, en función del resultado de la consulta se irán actualizando uno a uno los puertos.

Para que este método sea ejecutado cuando ocurre un cambio en el estado del switch, es necesario conectarlo con la señal UNIX enviada al terminar los casos de uso *LinkUp, LinkDown y AccesoSistema*. Al no ser una señal PyQt, no es válida la forma que se ha utilizado para casos de uso anteriores, por lo que es necesario

conectarlo utilizando la clase *signal* de Python, como se puede ver en la figura 4.14.

```
app =QtGui.QApplication(sys.argv)
myapp = Aplicacion()
#Función para manejar la recepción de una señal que indica
#que se ha cambiado el estado de un puerto
def receive_signal(*args):

    myapp.actualiza_interfaz()
    return
signal.signal(signal.SIGUSR1, receive_signal)
```

Figura 4.14: Conexión de señal UNIX con su *slot*.

4.2. Métodos relevantes

A lo largo del anterior apartado se ha hecho referencia a varios métodos que por sí solos no suponen la ejecución de un caso de uso, pero que, sin embargo, sirven de muleta para la realización de muchas tareas. Estos dos métodos son *cambia_vlan()* y *obtieneMAC()*, ambos presentes en la clase *gestionVlanSNMP*. Además de estos dos métodos, es necesario contar también con un método para realizar las operaciones SNMP `get`, `set` y `walk`. En este apartado no van a ser tratados ya que en la documentación de pysnmp se describe cómo realizar estas operaciones [Ily05].

4.2.1. Método obtieneMAC

Uno de los objetivos del presente Trabajo de Fin de Grado es poder conocer en todo momento qué dispositivos hay conectados a la red y para ello, conocer la dirección MAC de dichos dispositivos es fundamental. Esta información no viene reflejada en ninguna de las dos notificaciones SNMP que aluden al estado de un puerto. Es por ello que es necesario buscar una forma de obtener esta información. Este método recibe como parámetro el puerto del que se quiere obtener la dirección MAC. El primer problema es que la consulta no devuelve un resultado 'limpio'; es decir, no se trata de una matriz que contenga el número de puerto y la dirección MAC del dispositivo asociado. En su lugar, devuelve una matriz donde la primera columna de cada fila es el OID consultado seguido de la dirección MAC y la segunda columna es la VLAN a la que pertenece ese puerto. Unido a esto, hay que tener en

```
def obtieneMAC(self,puerto):
        #Hace la consulta SNMP para obtener la relación de puerto-mac que hay en el switch
        respuesta,valor = self.snmpwalk("1.3.6.1.2.1.17.7.1.2.2.1.2")
        lista = []
        for i in range(0,len(valor)):
            #Si el resultado que se está evaluando es el del puerto que queremos
            #obtener la MAC entra
            if(valor[i]==int(puerto)):

                #Separa la parte de la MAC de la parte del OID
                auxiliar  = str(respuesta[i]).split("1.3.6.1.2.1.17.7.1.2.2.1.2.")
                decimal = auxiliar[1].split(".")
                #Empieza en 1 porque el primer elemento corresponde a la vlan del puerto
                for i in range (1,len(decimal)):
                        #Guarda en una lista cada parte de la dirección MAC
                        #La función hex() devuelve el resultado de la forma "0x"
                        #Con la función replace lo eliminamos
                        if int(decimal[i]) > 15:

                            lista.append(hex(int(decimal[i])).replace("0x",""))
                        #Si el valor es menor de 15 (F), coloca un 0 delante para
                        #que cada octeto cuente con 2 dígitos
                        else:

                            lista.append("0" + hex(int(decimal[i])).replace("0x",""))
                return ":".join(lista)

        return None
```

Figura 4.15: Desarrollo del método ObtieneMAC.

cuenta que la dirección MAC no vendrá dada de forma hexadecimal, sino que sera una sucesión de enteros separados por puntos, donde cada entero representa dos dígitos de la dirección. Un ejemplo del formato en el que viene la dirección MAC dentro de la cadena puede ser el siguiente: *248.169.99.222.57.218*.

Cada entero será necesario convertirlo a hexadecimal para después concatenarlo y obtener una dirección MAC de la forma aa:bb:cc:dd:ee:ff. Para ello, se hará uso de la función de Python *hex()*, que convierte un entero en una cadena de texto con formato hexadecimal. Otro inconveniente es que, al hacer la conversión entero a entero, por cada 8 bits se añade el prefijo '0x' a la cadena, así que habrá que utilizar el método *replace()* para eliminar este prefijo. Además si el número entero es menor que 16, es decir, se puede codificar con un único dígito, habrá que añadir un 0 al principio de la cadena que representa el entero de 8 bits.

Por último, y como ya se ha mencionado anteriormente, habrá que concatenar las seis cadenas resultantes, separándolas por dos puntos, para obtener la dirección MAC en el formato con el cual se representa normalmente. Todo este proceso queda reflejado en la figura 4.15.

4.2.2. Método cambia_vlan

Este es, quizá, el método más importante de toda la aplicación, ya que se encarga de mover de VLAN a los dispositivos conectados al *switch*, por lo que sin él no sería posible tener separados a los usuarios en diferentes redes en función de su rol. Se encuentra en el *script gestionVlanSNMP* y es utilizado por todos los métodos descritos en los casos de uso que necesiten cambiar la VLAN en la que se encuentra un puerto.

Recibe como parámetros el puerto al que se le debe de aplicar el cambio, la VLAN en la que se encuentra el puerto y la VLAN a la que se quiere cambiar.

Con respecto al desarrollo del método, en primer lugar es necesario realizar una operación `get` sobre el OID 1.3.6.1.2.1.17.7.1.4.3.1.2, seguido de un punto y la VLAN que ha sido recibida como segundo parámetro. La consulta devolverá un mapa de bits que refleja la presencia, o no, de la VLAN en los puertos del *switch*, siendo el puerto 1 el bit más significativo. Una vez obtenido el resultado, es necesario convertirlo a entero, ya que se trata de un tipo 'Octect String', que no puede ser modificado directamente. El objetivo de esta consulta es modificar el resultado, para eliminar el bit correspondiente al puerto sobre el que estamos trabajando y para quitarlo de la VLAN. Para ello se realizará una operación *AND*, entre el valor actual del OID y una cadena de bits, de igual tamaño que el proporcionado por la operación `get`, cuyos bits estén todos a uno exceptuando el que haga referencia al puerto, que deberá estar a 0. Una vez realizada la operación, se realiza una operación `set` sobre el mismo OID, para realizar los cambios.

Una vez realizadas las operaciones anteriores, el puerto que se está modificando no pertenece a la VLAN, así que hay que realizar una serie de operaciones similares para que ahora pase a pertenecer a la VLAN que se ha pasado como tercer argumento del método. En esencia todo se hace de la misma forma, pero en lugar de realizar una operación AND, se realiza una operación *OR* entre el valor obtenido y una cadena de bits que están todos a 0 excepto el que corresponde al puerto, cuyo valor debe ser 1.

Para terminar, es necesario modificar también el nodo *dot1qPvid*, que contiene el identificador de VLAN del puerto, y asignarle el de la nueva VLAN. En la figura 4.16 se encuentra el desarrollo del método al completo.

Cuando se realiza un cambio de VLAN, es necesario mantener el puerto desactivado durante 10 segundos. Esto es necesario para forzar a la máquina conectada al puerto a solicitar de nuevo una dirección IP. Si no se hiciera esto, la máquina

```
def cambia_vlan(self,puerto,vlan,vlan_nueva):

        #OID para consultar el estado de la vlan en los puertos
        oid = "1.3.6.1.2.1.17.7.1.4.3.1.2." + str(vlan)
        #Obtiene el estado
        val = self.snmpget(oid)
        #Se queda solo con la parte importante (los primeros 8 bits que representan los puertos
        #nuestro switch
        #Obtiene el número de bits de la respuesta para poder operar
        num_bits = len(val) * 8
        #Concatena el resultado para trabajar con el
        r = val[0]
        for i in range(1,len(val)):
            r = r + val[i]

        #Convierte el tipo OctecString a entero para poder operar con el
        s = int(r.encode('hex'), 16)
        #En función del puerto que vaya a cambiar de vlan se realiza la resta para eliminar
        #la vlan del puerto en la variable dot1qVlanStaticEgressPorts
        s = s & ~(1<<(num_bits-int(puerto)))
        #Convierte a cadena hexadecial el entero y lo completa con 8 0s para que la operación
        #no falle
        nuevo_valor = hex(s)
        #Cambia la variable dot1qVlanStaticEgressPorts de la vlan para eliminar la vlan
        @del puerto
        self.snmpset(oid,OctetString(hexValue=nuevo_valor.replace("0x","")))

        # Ahora hay que hacer los mismos cambios pero para la nueva vlan

        oid = "1.3.6.1.2.1.17.7.1.4.3.1.2." + str(vlan_nueva)
        #Obtiene el estado
        val = self.snmpget(oid)
        #Se queda solo con la parte importante (los primeros 8 bits que representan los puertos
        #nuestro switch
        #Concatena el resultado para trabajar con el
        r = val[0]
        for i in range(1,len(val)):
            r = r + val[i]
        #Convierte el tipo OctecString a entero para poder operar con el
        s = int(r.encode('hex'), 16)
        #Añade el bit correspondiente al nuevo puerto
        s = s | (1<<(num_bits-int(puerto)))
        nuevo_valor = hex(s)
        #Cambia la variable dot1qVlanStaticEgressPorts de la vlan para añadir la vlan
        #al puerto
        self.snmpset(oid,OctetString(hexValue=nuevo_valor.replace("0x","")))

        #Modifica el oid para este caso
        oid = "1.3.6.1.2.1.17.7.1.4.5.1.1." + str(puerto)
        #Finalmente hay que cambiar al puerto de vlan modificando  dot1qPvid
        self.snmpset(oid,Gauge32(vlan_nueva))
```

Figura 4.16: Método cambia_vlan.

permanecería con su antigua dirección IP, que no corresponde con la de su actual red. Es por este motivo por el que es necesario el campo *puerto_manipulado* de la tabla `switch` de la base de datos. Cuando el puerto sea desactivado por este motivo, ese campo será puesto a 1, para indicar que se está manipulando el puerto y no se debe cambiar su estado al recibir las trampas. Estos diez segundos transcurridos es un tiempo orientativo, y está puesto a este valor porque se supone que en ese periodo de tiempo la máquina conectada se da cuenta de que ha perdido la conexión. Para mejorar este sistema, se ha explorado la opción de poner un *Lease Time* a la concesión de la dirección IP muy pequeño, pero la herramienta utilizada permite poner este tiempo a 2 minutos como mínimo.

4.3. Pruebas

Una vez finalizada la fase de implementación de la aplicación, es necesario dar paso a una fase de pruebas donde se evalúe si se han alcanzado los objetivos descritos en el primer capítulo y si se han cumplido todos los requisitos del sistema expuestos en el capítulo 4 del presente Trabajo de Fin de Grado. Para ello, se dedicará esta sección a mostrar los resultados obtenidos y comprobar si se han satisfecho todos los requisitos del sistema.

4.3.1. Cumplimiento de requisitos

En este apartado se va a revisar el cumplimiento de los requisitos del sistema, empezando con los requisitos funcionales, para terminar concluyendo el apartado tratando los requisitos no funcionales.

FRQ-001

Este requisito hace referencia a la capacidad del sistema para mostrar el estado actual de los puertos del *switch*.

Este requisito es uno de las principales motivaciones de la realización del presente Trabajo. Hay dos formas de conocer el estado del *switch*. La primera, y más intuitiva, es visualizar la ventana principal de la aplicación gráfica, donde aparece dibujado un *switch* con sus enlaces conectados a diferentes dispositivos, que muestra el estado del enlace. La segunda es a través de la opción 'Ver puertos' del menú de la aplicación. En la figura 4.17 se muestra la apariencia de la interfaz principal.

Para comprobar que este requisito se cumple se han llevado a cabo una serie de pruebas consistentes en provocar todos los eventos que pueden ocurrir en la red (conexión o desconexión de un dispositivo, autenticación de un usuario, cambios de VLAN, etc), concluyendo con que el requisito queda satisfecho.

FRQ-002

Este requisito cuenta con la siguiente redacción: «el sistema deberá permitir autenticarse a los clientes de la red para poder acceder».

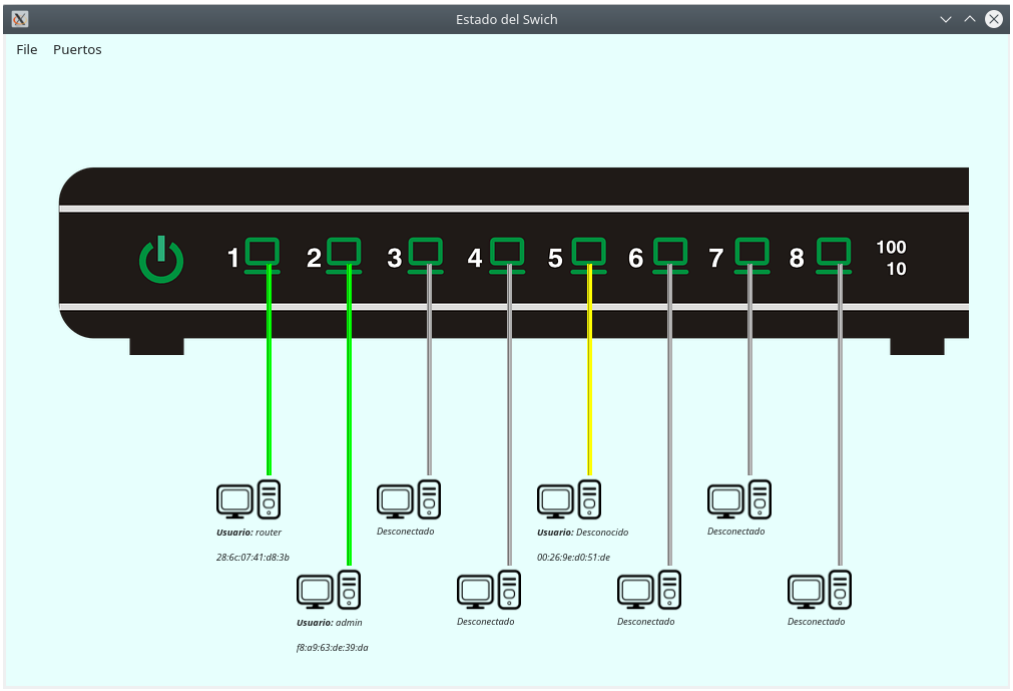


Figura 4.17: Aspecto de la ventana principal de la aplicación.

Para satisfacer este requisito se ha puesto en marcha una página web de acceso a la que serán redirigidos todos los usuarios que se encuentren sin identificar además de la creación de una serie de bloques de código para realizar las operaciones vistas en el caso de uso *AccesoSistema*.

Para realizar las pruebas, se ha procedido a autenticarse en la red con un dispositivo conectado al *switch* para comprobar si se realiza el cambio de VLAN y la dirección IP del dispositivo conectado cambia a una perteneciente a la de la red de usuarios autenticados. Las pruebas realizadas demostraron que así es, pero se comprobó que, una vez realizado el acceso, no se redirigía a una página distinta a la de acceso ni se mostraba ningún mensaje de éxito, por lo que el usuario podría no darse cuenta de que ya podía navegar libremente por la red. Para remediar esto se realizó una función con *JavaScript* para que muestre un mensaje en la página web cuando el usuario se autentique con éxito.

FRQ-003

Este requisito aborda la capacidad del sistema para gestionar la conexión de un puerto. Está estrechamente ligado al estado del *switch*, por lo que parte de las pruebas realizadas para el requisito FRQ-001 son válidas para este caso. De estas pruebas se puede extraer que cuando un nuevo dispositivo se conecta al *switch*, se realizan los cambios necesarios en la base de datos para que la interfaz gráfica refleje la situación actual del *switch*.

FRQ-004

Si el anterior requisito abordaba la capacidad de la aplicación para gestionar la conexión del un puerto del *switch*, este trata sobre la desconexión. Al igual que en el caso anterior las comprobaciones realizadas para el primer requisito son válidas en este caso.

FRQ-005

La descripción de este requisito menciona que «el sistema deberá ser capaz de cambiar la VLAN asociada a un puerto».

En la sección anterior se mostró el desarrollo del método *cambia_vlan*, que, sin satisfacer un caso de uso, sirve de apoyo para la realización de otros casos de uso, además de satisfacer este requisito. Para comprobar este requisito, al igual que en casos anteriores, sirven las pruebas realizadas para el primer requisito, aunque al tratarse de una función fundamental, se realizaron pruebas por separado para comprobar su buen funcionamiento y descartar errores. Estas pruebas consistieron en realizar diversos cambios de VLAN en diferentes puertos a través del método, sin necesidad de que ocurriera un cambio el el *switch*. Las pruebas arrojaron como resultado que el método funcionaba, pero era poco robusto debido a la forma de realizar el proceso. En lugar de utilizar operaciones lógicas, como se vio en el anterior apartado, la forma de cambiar el contenido del nodo *dot1qVlanStaticEgressPorts* era mediante sumas, es decir, al resultado obtenido mediante la operación `get`, se le sumaba o restaba un número entero que era igual al valor en decimal de la máscara de bits que se aplica ahora. En esencia el resultado es el mismo, pero era más propenso a dar problemas si no se seguía el flujo normal del programa. Con un ejemplo se entenderá mejor. Supongamos que tenemos un puerto del switch en la VLAN con identificador 100 y queremos cambiarlo a la VLAN con identificador

200. Si por algún fallo en la base de datos, el puerto ya se encuentra en la VLAN 200, con el método actual no ocurrirá nada, puesto que al realizar el cambio mediante operaciones lógicas solo se ve afectado el bit que representa al puerto que se está modificando. De la otra forma, como se trataba de sumas y restas de números enteros, podrían verse afectados otros puertos. En resumen, realizar un cambio de VLAN que no tenga sentido en un puerto, podría afectar al estado del resto de puestos, con la forma actual no.

Una vez realizado el cambio en la manera de modificar el PVID, se comprueba que ya no se produce ningún error y que el requisito se cumple.

FRQ-006

Este requisito aborda la necesidad de que el sistema tenga la capacidad para identificar un *host* conocido, cuando este se conecte al *switch*.

Para realizar las pruebas se ha añadido a la base de datos la dirección MAC de un dispositivo, además de un nombre para el dispositivo, y se le ha conectado al *switch* para comprobar en que VLAN se le incluye. Como era de esperar, tanto la interfaz gráfica como la base de datos reflejan que el puerto en el que ha sido conectado el dispositivo se encuentra en la VLAN de dispositivos conocidos. Esta circunstancia también puede ser comprobada en el *switch*, mediante SNMP o accediendo a la web de configuración.

FRQ-007

La descripción de este requisito dice así: «la interfaz gráfica deberá actualizarse cuando el estado del *switch* cambie».

Para realizar las pruebas del cumplimiento de este requisito nos valen las realizadas para el FRQ-001. Tras la realización de las mismas, se comprobó que cuando ocurre un evento que implique el cambio en uno de los puertos del *switch* , la interfaz gráfica se actualiza para reflejarlo, aunque se comprobó que debido a la forma que se ha elegido para enviar la señal que notifique el cambio, si hay varias instancias abiertas de la interfaz gráfica, el cambio solo aparecerá en la que se haya creado en último lugar. Para subsanar este error, se ha creado un cerrojo, que no permita abrir la interfaz gráfica si ya está abierta, incluyendo la opción de ejecución `-force`, para forzar la creación una nueva instancia de la interfaz gráfica.

FRQ-008

Este requisito hace referencia a la capacidad del sistema para proporcionar al administrador un mecanismo para modificar los datos de gestión que se almacenan en la base de datos.

Este requisito queda recogido y satisfecho en el caso de uso *CambiaValores*. A través del menú de la ventana principal se abre un cuadro de diálogo que muestra los parámetros a modificar y su valor actual. La forma de comprobar su funcionamiento es realizar diversos cambios en la base de datos a través de este menú, además de comprobar que los datos con los que se inicializan los cuadros de texto son los correctos. Tras realizar diferentes cambios a través de esta opción, se ha llegado a la conclusión de que funciona correctamente.

FRQ-009

La definición de este requisito dice así: «el sistema deberá contar con un mecanismo para actualizar la base de datos obteniendo la información directamente del *switch*».

Este requisito se ve satisfecho por el caso de uso *ActualizaBD*, que consiste en actualizar la base de datos con los datos obtenidos del *switch* mediante SNMP. Para revisar el cumplimiento de este requisito se han utilizado esta opción del menú modificando previamente algún registro de la base de datos para comprobar si se actualiza correctamente. Tras realizar varias pruebas se llega a la conclusión de que el mecanismo funciona, pero que en caso de haberse producido alguna modificación, esta no se ve reflejada en la interfaz. Para subsanar este error se llama al método *actualiza_interfaz()* después de actualizar la base de datos.

FRQ-010

Este requisito menciona la capacidad del sistema para mostrar de forma detallada el estado de los puertos. Tras realizar varias pruebas y cambios en el estado del *switch*, se comprueba que la información arrojada por la tabla generada a partir de la elección de la opción 'Ver puertos', es consistente y además de coincidir con los datos de la base de datos, muestra de forma fidedigna el estado de los puertos del *switch*.

NFRQ-001

«El sistema deberá permitir refrescar la interfaz gráfica».

Este requisito queda satisfecho por la opción 'Actualizar interfaz' del menú de la ventana principal. Se utiliza el mismo método que cuando se refresca la interfaz debido a un cambio en el estado del *switch*, por lo que se puede concluir que el requisito queda satisfecho.

NFRQ-002

Este requisito impone la necesidad de que la versión de SNMP a utilizar sea la versión SNMPv2c.

El cumplimiento de este requisito queda demostrado a lo largo de la presente memoria, puesto que se usan *communities* como método de autenticación para acceder a la base de datos de información gestionada del *switch* .

4.3.2. Comparativa de los recursos utilizados por la aplicación frente a Packetfence

Como ya se mencionó en el primer capítulo de la presente memoria, el rendimiento y los recursos utilizados por las principales soluciones comerciales existentes son uno de las principales motivaciones del presente Trabajo de Fin de Grado. Es por ello que se va a dedicar esta sección del documento a detallar los resultados obtenidos tras la medición del consumo de la herramienta desarrollada y Packetfence, puesto que el rendimiento de la aplicación desarrollada es discriminante con respecto al de Packetfence. Se ha elegido esta herramienta por ser las más completa y la que, entre sus otras muchas características, realiza un control de acceso a red bastante similar al realizado en este trabajo.

Para poder realizar esta comparativa, se ha instalado la distribución Packenfence ZEN 9.3.0 en un máquina virtual para la realización de las pruebas pertinentes.

El primer aspecto a tener en cuenta es el espacio que ocupa en el disco cada opción. Mientras que el conjunto de *scripts* Python utilizados, unidos a la base de datos utilizada para guardar la información del *switch*, suman un total de 7.64 MB, la distribución de Packetfence ocupa 3.2 GB. Además de estos 3.2 GB, la máquina

virtual reserva hasta 40 GB de espacio de disco para la distribución.

En segundo lugar, es necesario realizar la comparativa entre la memoria RAM utilizada por ambas opciones. Para ello, se ha hecho uso del comando Linux `top`, que, además de otros datos, muestra la memoria RAM utilizada por cada proceso en tiempo real. Mientras que la aplicación desarrollada en el presente trabajo tiene un consumo de RAM prácticamente residual, ya que oscila entre los 40 y los 80 MB, dependiendo de la operación que este realizando la herramienta, siendo la operación de inicializar la base de datos con la información del *switch* la operación más costosa. En cuanto a la memoria consumida por Packetfence, oscila entre 2,5 y los 5,6 GB de RAM, un consumo notablemente mayor. La utilización media de la memoria RAM es de 45 MB en la herramienta desarrollada en este Trabajo, mientras que Packetfence hace un uso medio de la memoria RAM de 3.4 GB.

Para finalizar con este apartado, es necesario mencionar el consumo de CPU consumido por ambas opciones. Este es el apartado que menos diferencia hay entre las mediciones, ya que los datos obtenidos son muy parejos, aunque cabe destacar que el consumo de CPU por parte de Packetfence es en torno a un 30 % mayor y se producen picos de utilización mucho mayores que en la aplicación desarrollada, llegando en algunos casos al 90 % de utilización.

Capítulo 5

Manual para el administrador

Para poner en marcha el mecanismo de control de acceso a red que motiva la realización del presente Trabajo de Fin de Grado, no basta solo con la parte *software* que ha sido desarrollada en el anterior capítulo. Todo lo expuesto anteriormente no sirve de nada si no se le proporciona una dirección IP a los dispositivos conectados al *switch*.

El objetivo de este capítulo es hacer hincapié en la serie de servicios que subyacen a la aplicación y que son igual de importantes para el buen funcionamiento de la herramienta. Para ello, se va a hablar de los diferentes servicios que hay que poner en marcha y configurar, además de proporcionar una explicación de por qué son necesarios.

5.1. Configuración del *switch*

Antes de empezar a hablar sobre la configuración de los distintos servicios que dan soporte a la herramienta, es necesario mencionar que hay que configurar el *switch* utilizado en la red. Esta configuración no es más que la creación de las redes virtuales que van a conformar la red y su asignación a cada puerto. La configuración dependerá del modelo utilizado, por lo que no se va a profundizar en este asunto.

Como ya es sabido (véase la sección 4.1), el *switch* debe tener configuradas al menos tres VLAN, que corresponden a las VLAN restringida, a la VLAN de usuarios registrados y a la de dispositivos conocidos. Para el desarrollo del Trabajo,

se ha dejado los puertos 1 y 2 del *switch* como reservados, puesto que es donde irán conectados el *router* y la máquina que corra la aplicación, respectivamente. Estos puertos no serán movidos de VLAN aunque su estado (conectado y desconectado) cambie. Además, deberán tener asociadas las tres VLANs como *tagged*, además de la VLAN de gestión del *switch*, que será *tagged* también. En un principio, el resto de puertos solo tendrá asociada una VLAN, que será la restringida, y será configurada como *untagged*.

5.2. Servidor DNS y DHCP

Estos dos servicios se encargan de proporcionar los parámetros de red a los usuarios conectados al *switch*. Mientras que el servidor DHCP se encarga de proporcionar una dirección IP, además de proporcionar parámetros como la dirección del servidor DNS o el *router*, el servidor DNS se encarga de atender las peticiones de resolución de nombre de dominio.

El motivo de que ambos servidores se encuentren en el mismo apartado no es otro que la utilización de *dnsmasq* para proporcionar ambos servicios. Se ha elegido *dnsmasq* por dos motivos: está diseñado para ser liviano y consumir poco, característica que va acorde con una de las motivaciones de este trabajo. El segundo motivo es que proporciona un mecanismo para realizar la redirección de dirección IP muy sencillo mediante la opción de configuración `address` [Sim20].

Esta solución no está exenta de problemas, pues si bien es cierto que con la opción mencionada anteriormente será posible redirigir todas las consultas DNS a la dirección del servidor web que contiene la página web de acceso, esta opción es aplicada de forma genérica para todas las redes a las que sirva como servidor de nombre de dominio. Como las consultas realizadas desde las VLAN de usuarios autenticados y dispositivos conocidos sí que deben ser resueltas correctamente, es necesario tener corriendo dos servicios de forma simultanea, uno para la red restringida y otro para el resto.

Para poder tener en marcha los dos servicios a la vez es necesario crear un nuevo servicio de `systemd`. Para ello, lo primero que se debe hacer es crear un fichero en el directorio */etc/systemd/system*, que es el directorio que contiene los servicios creados por el usuario. En la figura 5.1 se muestra como debe ser este fichero. Al inicio del fichero se establecen los requerimientos necesarios para que el servicio pueda ser puesto en marcha. La opción *ExecStartPre* define las tareas a realizar antes de la puesta en marcha del servicio, mientras que la opción *ExecStart* pone en

```
Description=dnsmasq - A lightweight DHCP and caching DNS server
Requires=network.target
Wants=nss-lookup.target
Before=nss-lookup.target
After=network.target

[Service]
Type=forking
PIDFile=/run/dnsmasq.pid

# Test the config file and refuse starting if it is not valid.
ExecStartPre=/usr/sbin/dnsmasq --test

ExecStart=/usr/sbin/dnsmasq --interface=enp1s0f1.300 --except-interface=lo
    --bind-interfaces --dhcp-range=192.168.3.50,192.168.3.150,12h
    --address=/#/192.168.3.10 --domain=restringido.tfg.es,192.168.3.0/24
    --dhcp-option=6,192.168.3.10 --dhcp-option=3,192.168.31.1 --domain-needed


ExecStop=/bin/kill -HUP $MAINPID

ExecReload=/bin/kill -HUP $MAINPID

[Install]
WantedBy=multi-user.target
```

Figura 5.1: Fichero de creación del servicio dnsmasq.

marcha el servicio. En este caso, para ilustrarlo mejor, aparecen todas las opciones en el comando, aunque también se pueden guardar las opciones de configuración del servidor en un fichero y utilizar la opción `-C` del comando `dnmasq`. Por último, se establece qué se debe hacer en caso de parada o reinicio del servicio. Para iniciar el servicio cuando se inicie el sistema es necesario ejecutar la orden `systemctl enable <servicio>`, para los dos servicios creados [Deb20].

Una vez mencionada la forma de poner en marcha un nuevo servicio en `systemd`, es necesario abordar la descripción de las características que tiene que tener cada uno de estos dos servicios. Ambos son muy similares, con la única diferencia de que el servicio que escuche en la red restringida tendrá la opción `address` para redirigir las peticiones. Con esta opción es posible redirigir las consultas DNS sobre un determinado nombre de dominio siempre a la misma dirección IP. Por lo demás son similares: ambos cuentan con una opción para establecer en rango de direcciones IP que debe ofrecer el servidor, el nombre de dominio para cada red, el servidor DNS para cada red y el router de cada red, además de establecer en que interfaz debe escuchar. En las figuras 5.3 y 5.2 se puede ver como debería quedar la configuración de los dos servicios.

Como ya se ha comentado anteriormente, la opción `address` responde a las peticiones sobre un determinado nombre de dominio a la dirección IP proporcionada. Si, como se puede ver en la figura 5.3, el valor proporcionado es #, significa que todas las peticiones que lleguen al servidor deben ser respondidas con la misma IP.

```
#Fichero de configuración de dnsmasq

except-interface=enp1s0f1.300
bind-interfaces
domain-needed

dhcp-range=192.168.2.50,192.168.2.50,12h
domain=conocidos.tfg.es,192.168.2.0/24
dhcp-option=tag:enp1s0f1.200,6,192.168.2.10


dhcp-range=192.168.1.50,192.168.1.50,12h
domain=conocidos.tfg.es,192.168.1.0/24
dhcp-option=6tag:enp1s0f1.100,192.168.1.10

dhcp-option=3,192.168.31.1
```

Figura 5.2: Configuración de dnsmasq para el resto de redes.

```
#Fichero de configuración de dnsmasq

interface= enp1s0f1.300
#except-interface=lo
domain-needed

dhcp-range=192.168.3.50,192.168.3.150,12h
domain=restringido.tfg.es,192.168.3.0/24

dhcp-option=3,192.168.31.1

dhcp-option=6,192.168.3.10

address=/#/192.168.3.10
```

Figura 5.3: Configuración de dnsmasq para la VLAN restringida.

La opción `interface` indica en qué interfaz del equipo debe escuchar el servicio, mientras que la opción `except-interface` indica en que interfaces no debe escuchar. La opción `dhcp-range` indica el rango de direcciones IP que pueden ser entregadas mediante DHCP, incluyendo la etiqueta `tag:nombre_interfaz` si se quiere que un determinado rango de direcciones sea proporcionado solamente a las peticiones recibidas por una interfaz en concreto. La opción `domain` establece el nombre de dominio para una determinada red y la opción `dhcp-option` sirve para indicar la dirección del router y del servidor DNS, con los valores 3 y 6 respectivamente.

5.3. Servidor snmptrapd

Como el funcionamiento del sistema se basa en la recepción de trampas enviadas por el agente SNMP cuando un evento ocurre en el *switch*, es necesario contar con un mecanismo que sea capaz de recibir e interpretar las trampas. Para realizar esta tarea se ha utilizado el servicio *snmptrapd* que forma parte del paquete *Net-SNMP* y que no es más que un demonio que escucha en el puerto UDP 162 a la espera de recibir trampas. Además, para que el manejo de las trampas sea más sencillo se ha hecho uso del paquete snmptt *(SNMP Trap Translator)*, que es un traductor de trampas utilizado con *snmptrapd* de forma conjunta. De esta forma, será el demonio *snmptrapd* quien se encargue de recibir las trampas, pero de su procesado se encargará *snmptt* [Deb14, Ale13].

En primer lugar hay que instalar ambos paquetes, si no estuvieran instalados ya mediante las siguientes órdenes (validas para una distribución GNU/Linux basada en *Debian*):

```
apt-get install snmptrapd
apt-get install snmptt
```

Una vez instalados, es necesario indicar que el demonio *snmptrapd* se arranque cuando se encienda el ordenador, para ello, al igual que antes, es necesario usar el comando: `systemctl enable snmptrapd`.

En el fichero de configuración del demonio, que se encuentra en */etc/snmp/snmptrapd.conf* es necesario indicar que para cualquier trampa que reciba, delegue su gestión al demonio *snmptt* mediante la opción: `traphandle default`

```
EVENT IF-MIB::linkDown .1.3.6.1.6.3.1.1.5.3 "Status Events" Normal
FORMAT Link down on interface $1. Admin state: $2. Operational state: $3
EXEC python /home/dario/TFG/gestionVlanSNMP.py down $1
SDESC
A linkDown trap signifies that the SNMP entity, acting in
an agent role, has detected that the ifOperStatus object for
one of its communication links is about to enter the down
state from some other state (but not from the notPresent
state). This other state is indicated by the included value
of ifOperStatus.
EDESC
#
#
#
EVENT linkUp .1.3.6.1.6.3.1.1.5.4 "Status Events" Normal
FORMAT Link up on interface $1. Admin state: $2. Operational state: $3
EXEC python /home/dario/TFG/gestionVlanSNMP.py up $1
SDESC
A linkUp trap signifies that the SNMP entity, acting in an
agent role, has detected that the ifOperStatus object for
one of its communication links left the down state and
transitioned into some other state (but not into the
notPresent state). This other state is indicated by the
included value of ifOperStatus.
EDESC
#
#
```

Figura 5.4: Fichero de configuración snmptt.conf.

`/usr/sbin/snmptt`. En el fichero de configuración de este segundo servicio, que se encuentra en */etc/snmp/snmptt.ini*, se debe dejar la configuración tal y como está, asegurándose de que el modo es 'standalone' y no 'daemon'. Por último, en el fichero */etc/snmp/snmptt.conf*, es necesario crear una entrada para los dos tipos de trampa que interesan para el Trabajo, como se puede ver en la figura 5.4, donde el aspecto más importante es la opción *EXEC*, que ejecuta el fichero encargado de la gestión de la red, pasándole el estado del puerto y su número.

5.4. Servidor web

Es necesario contar con un servidor web que aloje la página de acceso a la red. Para ello es necesario instalar un servidor web en la máquina que contenga la aplicación. Como el servidor no necesita ningún tipo de configuración compleja, con la instalación del paquete que lo contiene, la página web ya funcionará. Es por ello que en esta sección se va a explicar las operaciones a realizar desde que el usuario pulsa el botón 'Acceder' de la web, hasta que se llama al *script gestionVlanSNMP.py* para que realice el cambio de VLAN en el puerto.

```
//Función que se encarga de obtener el puerto y actualizar la base de datos
function acceso($basedatos, $login){
    //Obtiene la IP del cliente
    $ipAddress=$_SERVER['REMOTE_ADDR'];
    $macAddr=false;

    #Ejecuta el comando para obtener la dirección mac
    $arp=` /usr/bin/arp -a $ipAddress`;
    $lines=explode("\n", $arp);
    #echo $ipAddress;

    #look for the output line describing our IP address
    //Separa los campos devueltos por el comando. El 1 y el 3 serán IP y MAC respectivamente
    $cols=preg_split('/\s+/', trim($lines[0]));
    //Evalúa si la IP obtenida es la misma que la del cliente
    if(preg_replace("/\(|\)/", "", $cols[1])==$ipAddress)
    {
        //Asigna la MAC
        $macAddr= $cols[3];
        //Consulta para obtener el puerto asociado a esa MAC
        $sql="SELECT puerto FROM switch WHERE mac='$macAddr'";
        //Ejecuta la consulta
        $query = $basedatos->query($sql);
        if (!$query){
                echo"No se puede ejecutar la consulta.";
        }

        $row = $query->fetchArray();
        $puerto = $row[0];
        //Actualiza la base de datos
        $basedatos->exec("UPDATE switch SET vlan=100, aux=1, login='$login' where puerto
         = $puerto");
        //Cierrra la base de datos
        $basedatos->close();

        //Ejecuta el comando
        `python /home/dario/TFG/gestionVlanSNMP.py login $puerto`;
    }
}
?>
```

Figura 5.5: Obtención de dirección MAC y puerto.

En primer lugar, es necesario comentar que la página web ha sido desarrollada en el lenguaje PHP. La web de acceso no es más que un formulario con un campo para introducir el *login*, un campo para introducir la contraseña, y un botón para acceder con los datos introducidos. Cabe mencionar también que, si alguno de los dos campos está vacío, no dejará continuar y avisará de ello. Una vez el usuario pulsa el botón 'Acceder' los datos introducidos se mandan mediante el método `POST` de PHP al script *acceso.php*.

En este *script*, lo primero que se hace es consultar en la base de datos si las credenciales son correctas. De ser así, obtiene la dirección IP del usuario mediante la línea `_SERVER 'REMOTE_ADDR'`. Con la dirección IP ejecuta el comando `arp -a ipAddress` para obtener la dirección MAC del dispositivo conectado. Una vez se ha obtenido la dirección MAC ya es posible conocer a que puerto está conectado, mediante una consulta a la base de datos. Por último, llama al *script gestionVlanSNMP.py* para cambiar al puerto de VLAN. En la figura 5.5 se muestra como realizar lo expuesto anteriormente.

```
auto enp1s0f1.200
iface enp1s0f1.200 inet static
        address 192.168.2.10
        netmask 255.255.255.0
```

Figura 5.6: Ejemplo de definición de interfaz en el fichero /etc/network/interfaces.

5.5. Interfaces de la máquina

Esta sección pone punto y final al presente capítulo. En ella se va a hablar de la configuración de las interfaces de la máquina que corre la aplicación. Ante la presencia de varias redes virtuales, es necesario crear una interfaz de red virtual para cada VLAN. Habrá que crear un total de cuatro interfaces virtuales sobre la interfaz física, una para cada VLAN y una a mayores para la VLAN de gestión del *switch*.

Para crear las diferentes interfaces virtuales hay dos opciones. En primer lugar, se pueden crear una a una con el comando `ip link` y darles una dirección IP con el comando `ifconfig`. Esta solución es temporal, ya que debe hacerse cada vez que la máquina se encienda, por lo que queda descartada. La segunda opción es crear un fichero donde se establezcan las interfaces del equipo y su tipo. Esta es, generalmente, la mejor solución ya que es permanente. El fichero en cuestión se ubica en */etc/network/interfaces*. Este fichero será leído cada vez que el equipo arranque y se crearán las interfaces virtuales. En la figura 5.6 se muestra un ejemplo de configuración de una interfaz en dicho fichero. Con la palabra clave `auto` se indica que la interfaz debe ser activada siempre, mientras que el propio nombre de la interfaz indica que se trata de una interfaz virtual con VID igual a 200, ya que el nombre de la interfaz está compuesto del nombre de la interfaz física, seguido de un punto y el VID.

Capítulo 6

Conclusiones y líneas futuras

El sexto y último capítulo sirve para poner el punto y final a la presente Memoria. Este capítulo está dividido en dos secciones que abordarán las conclusiones extraídas durante la realización del TFG, y un conjunto de líneas futuras por donde podría avanzar el trabajo si se deseara continuar con él algún día.

6.1. Conclusiones

Ante la necesidad de implantar un mecanismo para el control de acceso a red sin tener que depender de opciones comerciales, que generalmente suelen implicar un alto consumo de recursos computacionales, se decidió poner el marcha el presente TFG. A lo largo de la presente memoria, que está próxima a acabar, el lector ha podido conocer la motivación de la realización de este TFG, además de conocer los objetivos que persigue. Por otra parte, también ha servido para que el lector pueda conocer las tecnologías base de la herramienta y para comprobar que en el mundo de Internet, las cosas no se dan como se espera en un principio, teniendo el ejemplo del protocolo SNMP, que nació como una solución temporal hasta la migración a redes OSI y que a día de hoy es el protocolo de gestión de red más utilizado.

A lo largo del documento se ha hecho hincapié en que el trabajo no se basa solo en una aplicación *software* que gestiona la VLAN de cada puerto y muestra su estado de forma gráfica, sino que subyacen una serie de servicios y tecnologías que hacen posible haber conseguido los objetivos marcados en el primer capítulo.

Se ha comprobado que la herramienta desarrollada cumple con una de sus prin-

cipales motivaciones, ya que el consumo de recursos computacionales es mucho menor que el de la opción comercial utilizada para comparar, Packetfence. El espacio en disco utilizado por Packetfence es del orden de 700 veces mayor que el que ocupa la herramienta desarrollada y una utilización media de la memoria RAM es 75 veces mayor. En cuanto a la utilización de la CPU, varía bastante, sobretodo con Packetfence, pero si se puede comprobar que de media, el consumo por parte de Packetfence es en torno a un 30 % mayor.

Se ha demostrado que la decisión de utilizar dnsmasq como servidor DNS y DHCP ha sido buena, ya que ha permitido la redirección a la página web de acceso de manera sencilla y de forma exitosa. Además, el hecho de tener dos procesos de dnsmasq corriendo a la vez no ha supuesto ninguna desventaja en términos de rendimiento, ya que si se hubiera optado por otras alternativas habría sido necesario tener ambos servicios por separado y el número de procesos sería el mismo. Las mismas conclusiones se pueden sacar de la utilización de SQLite como gestor de base de datos. En términos de rendimiento ha supuesto una gran ventaja, ya que al ser un gestor *serverless* se evita tener un proceso adicional activo, mientras que no ha supuesto ninguna desventaja a la hora de trabajar, ya que las consultas y operaciones que se realizan en la base de datos son bastante sencillas y no es necesario un gestor de base de datos más potente.

En lo personal, este Trabajo me ha servido para aprender un nuevo lenguaje de programación, como es *Python*, además de haber realizado la interfaz gráfica, aspecto en el cual mi experiencia era escasa. También ha servido para entrar en contacto con varias herramientas a la vez, aunque sus orígenes no tengan nada que ver, y a enfrentarme a los problemas que iban surgiendo, buscando una manera de solucionarlos.

Por último, es necesario mencionar que aunque los objetivos marcados al inicio del documento han sido alcanzados y se ha conseguido construir una herramienta para el control de acceso a red completamente funcional y que puede operar consumiendo muy pocos recurso, aún hay mucho camino por explorar, puesto que hay aspectos de la herramienta que pueden ser mejorados y se pueden incluir nuevas funcionalidades.

6.2. Líneas de trabajo futuro

La herramienta desarrollada en este Trabajo tiene muchos caminos hacia donde mirar para su evolución. El hecho de ser una aplicación desarrollada en *Python*,

hace que tenga todo a favor para evolucionar, puesto que es uno de los lenguajes de programación más utilizados a día de hoy, con constantes actualizaciones de sus paquetes y librerías.

Uno de los aspectos a mejorar de la herramienta es su ventana principal. Los elementos que la conforman están dispuestos siguiendo un posicionamiento absoluto. Esta circunstancia hace que a los elementos que forman la interfaz, como el *switch* o los ordenadores, no se le puedan añadir barras de desplazamiento. Al no poder añadirse este elemento a la interfaz, si las dimensiones de los *widgets* superan las del tamaño de la ventana, no se podrá hacer *scroll* para poder visualizar los que queden más a la derecha. Por este motivo, en la ventana principal solo se muestran los primeros ocho puertos que tenga el *switch*, siendo necesario acudir a la opción 'Ver puertos' para conocer el estado del *switch* al completo. Cabe mencionar que aunque el número de puertos del *switch* exceda de ocho, la aplicación sigue realizando el control de acceso de forma correcta. Para subsanar este error se han intentado varias opciones como disminuir el tamaño de los elementos de la ventana en función del número de puertos para que entren todos en la ventana o disponer los puertos del *switch* de otra forma, pero ninguna ha resultado satisfactoria.

En caso de continuar con el desarrollo de la herramienta, sería interesante introducir nuevas funcionalidades. La base de datos de gestión de SNMP es muy extensa, por lo que se podrían monitorizar otros parámetros de la red mediante este protocolo. Otra opción interesante sería establecer un registro en el que guardar el momento en el que se han producido conexiones o desconexiones en el *switch*, incluyendo datos como la hora en la que se produce el evento, el puerto, la dirección MAC del dispositivo y su *login*, si es que lo tuviera. Además, sería bueno preparar la herramienta para que pueda realizar la autenticación SNMP mediante un mecanismo distinto a la comunidades, por si el *switch* utilizado solo soportara otro tipo de autenticación, y mejorar el método encargado de cambiar los parámetros del *switch* para que no solo sea una modificación de la base de datos, sino que estos cambios afecten también al *switch*.

Bibliografía

[Ale13] Alex Burger. Página del manual de systemd.unit. `systemd.unit(8)`, 2013.

[Blu02] U. Blumenthal y B. Wijnen. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3). RFC 3414 (INTERNET STANDARD), diciembre de 2002. Updated by RFC 5590.

[Cas88] J. Case, M. Fedor, M. Schoffstall y J. Davin. Simple Network Management Protocol. RFC 1067, agosto de 1988. Obsoleted by RFC 1098.

[Cas89] J. Case, M. Fedor, M. Schoffstall y J. Davin. Simple Network Management Protocol (SNMP). RFC 1098, abril de 1989. Obsoleted by RFC 1157.

[Cas90] J. Case, M. Fedor, M. Schoffstall y J. Davin. Simple Network Management Protocol (SNMP). RFC 1157 (Historic), mayo de 1990.

[Cas93a] J. Case, K. McCloghrie, M. Rose y S. Waldbusser. Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1448 (Proposed Standard), abril de 1993. Obsoleted by RFC 1905.

[Cas93b] J. Case, K. McCloghrie, M. Rose y S. Waldbusser. Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1442 (Proposed Standard), abril de 1993. Obsoleted by RFC 1902.

[Cer88] V. Cerf. IAB recommendations for the development of Internet network management standards. RFC 1052, abril de 1988.

[Dav87] J. Davin, J. Case, M. Fedor y M. Schoffstall. Simple Gateway Monitoring Protocol. RFC 1028 (Historic), noviembre de 1987.

[Deb14] Debian GNU/Linux. Página del manual de snmptrapd.conf. `snmptrapd.conf(5)`, 2014.

[Deb20] Debian GNU/Linux. Página del manual de systemd.unit. `systemd.unit(5)`, 2020.

[Gal93] J. Galvin y K. McCloghrie. Security Protocols for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1446 (Historic), abril de 1993.

[Goo16] Google. How people use their devices. `https://www.thinkwithgoogle.com/intl/en-aunz/advertising-channels/mobile/device-use-marketer-tips/`, 2016. Visitada por última vez en Enero de 2020.

[Har02] D. Harrington, R. Presuhn y B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411 (INTERNET STANDARD), diciembre de 2002. Updated by RFCs 5343, 5590.

[IEE90] IEEE. IEEE Standard Glosary of Software Engineering Terminology. 1990.

[IET06] IETF Bridge MIB Working Group. Definición de Q-BRIDGE-MIB, 2006.

[Ily05] Ilya Etingof. PySNMP Documentation. `http://snmplabs.com/pysnmp/docs/tutorial.html`, 2005. Visitada por última vez en Enero de 2020.

[Ily19] Ilya Etingof. PySNMP, SNMP History. `http://snmplabs.com/pysnmp/docs/snmp-history.html`, 2019. Visitada por última vez en Enero de 2020.

[Ins] Insitute of Electrical and Electronics Engineers, Inc. 802 ieee standard for local and metropolitan area networks: Overview and architecture. `http://standards.ieee.org/getieee802/download/802-2001.pdf`. Febrero 2014.

[Inv19] Inverse Inc. Installation Guide for Packetfence 9.0. `https://packetfence.org/doc/PacketFence_Installation_Guide.html`, 2019. Visitada por última vez en Enero de 2020.

[Inv20] Inverse Inc. Packetfence 9.0. `https://packetfence.org/`, 2020. Visitada por última vez en Enero de 2020.

[Joh00] M. S. Johns. Diffie-Helman USM Key Management Information Base and Textual Convention. RFC 2786 (Experimental), marzo de 2000.

[Kee00] H. Keen. IEEE 802.1q: Virtual bridged local area networks. *IEEE Network*, volumen 14, págs. 3–3, 07 de 2000.

[Lam06] T. Lammle. *CCNA Cisco Certified Network Associate Study Guide Deluxe Edition*. SYBEX Inc., Alameda, CA, USA, 6ª edición, 2006.

[Mau05] D. R. Mauro y K. J. Schmidt. *Essential SNMP*. O'Reilly Media, segunda edición, 2005.

[McC88a] K. McCloghrie y M. Rose. Management Information Base for network management of TCP/IP-based internets. RFC 1066, agosto de 1988. Obsoleted by RFC 1156.

[McC88b] K. McCloghrie y M. Rose. Structure and identification of management information for TCP/IP-based internets. RFC 1065 (INTERNET STANDARD), agosto de 1988. Obsoleted by RFC 1155.

[McC91] K. McCloghrie y M. Rose. Management Information Base for Network Management of TCP/IP-based internets:MIB-II. RFC 1213 (INTERNET STANDARD), marzo de 1991. Updated by RFCs 2011, 2012, 2013.

[Nag20] Nagios Enterprise. Nagios. `https://www.nagios.com`, 2020. Visitada por última vez en Enero de 2020.

[Par87] C. Partridge y G. Trewitt. High-level Entity Management System (HEMS). RFC 1021 (Historic), octubre de 1987.

[Pre10] R. S. Pressman. *Ingeniería del software. Un enfoque práctico*. McGraw-Hill, 7ª edición, 2010.

[Pri88] P. Prindeville. BOOTP vendor information extensions. RFC 1048, febrero de 1988. Obsoleted by RFCs 1084, 1395, 1497, 1533.

[PyQ14] PyQt, SourceForge. PyQt4 Reference Guide. `https://www.riverbankcomputing.com/static/Docs/PyQt4/`, 2014. Visitada por última vez en Enero de 2020.

[Ros81] E. Rosen. Vulnerabilities of network control protocols: An example. RFC 789, julio de 1981.

[Ros90] M. Rose, K. McCloghrie y J. Davin. Bulk Table Retrieval with the SNMP. RFC 1187 (Experimental), octubre de 1990.

[Sim20] Simon Kelley. Página del manual de dnsmasq. `dnsmasq(8)`, 2020. Visitada por última vez en Enero de 2020.

[Sin02] W. D. Sincoskie. Broadband packet switching: A personal perspective. *Comm. Mag.*, volumen 40, nº 7, págs. 54–66, julio de 2002.

[Som10] I. Sommerville. *Software Engineering.* Addison-Wesley Publishing Company, USA, 9ª edición, 2010.

[Sta98] W. Stallings. *SNMP,SNMPV2,Snmpv3,and RMON 1 and 2.* Addison-Wesley Longman Publishing Co., Inc., USA, tercera edición, 1998.

[Sta06] W. Stallings. *Data and Computer Communications (8th Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[Sum07] M. Summerfield. *Rapid Gui Programming with Python and Qt: The Definitive Guide to Pyqt Programming.* Prentice Hall Press, USA, primera edición, 2007.

[Wij02] B. Wijnen, R. Presuhn y K. McCloghrie. View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP). RFC 3415 (INTERNET STANDARD), diciembre de 2002.

[Wil88] M. Willet y R. Martin. LAN management in an IBM framework,. *IEEE Network: The Magazine of Global Internetworking*, Marzo de 1988.