Trabajo de Fin de Grado

Aplicación de SDN para el control del tráfico de red en base a usuarios

Using SDN for user based network traffic control .

Guillermo Vicente Sánchez Guindulain

La Laguna, 4 de marzo de 2017

D. **Jonas Philipp Lüke**, con N.I.E. X0581666-L profesor Ayudante Doctor adscrito al Departamento de Ingeniería Industrial de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Aplicación de SDN para el control del tráfico de red en base a usuarios.* "

ha sido realizada bajo su dirección por D. **Guillermo Vicente Sánchez Guindulain**, con N.I.F. 42.222.883-N

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de marzo de 2017

Agradecimientos

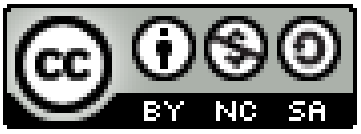
En primer lugar me gustaría dar las gracias a mi tutor Jonás Philipp Lüke, por su paciencia y sus buenas maneras en el transcurso del desarrollo del TFG, sin él no hubiera sido posible llevarlo acabo.

En segundo lugar a los compañeros y profesores que he tenido a lo largo de la carrera.

Y en último lugar y por ello no menos importante, a mi familia y amigos más cercanos que me han ayudado en los momentos más difíciles.

Un abrazo a todos y muchas gracias.

Licencia



Resumen

El objetivo de este trabajo ha sido conocer la tecnología SDN, aprender como funciona el protocolo OpenFlow y desarrollar un caso de uso basado en este paradigma. El caso de uso ha sido el control del tráfico de red en base a distintos grupos de usuarios.

Se ha desarrollado un controlador de permisos en Ryu, un autenticador web y se ha trabajado con topologías creadas mediante Mininet.

Se han manejado los protocolos de red necesarios para controlar el tráfico de una red plana, DHCP y ARP mediante módulos implementados en Ryu, el tráfico broadcast se ha controlado en el controlador de permisos.

Se ha implementado una aplicación de switching básica y otra más compleja para controlar el envío de paquetes de red.

También, se han gestionado las credenciales y permisos de usuarios/grupos almacenándolos en una base de datos.

Finalmente, se han realizado pruebas para comprobar el correcto funcionamiento de las aplicaciones implementadas.

Palabras clave: SDN, OpenFlow, redes de computadores, internet, gestión de red, protocolos de red.

Abstract

The goal of this work has been to know SDN technology, to learn how the OpenFlow protocol works and to develop a use case based on this paradigm. The use case has been the control of network traffic based on different groups of users.

A permissions handler has been developed using the Ryu controller, a web authenticator was developed and different topologies have been created using Mininet.

The network protocols required to control the traffic of a flat network, DHCP and ARP have been handled using modules implemented with Ryu, broadcast traffic has been controlled in the permissions controller.

A basic and a more complex switching application has been implemented to control the sending of network packets.

Credentials and user / group permissions have been managed and stored in a database.

Finally, tests have been performed to verify the correct operation of the implemented applications.

Keywords: *SDN, OpenFlow, computer networks, internet, network management, network protocols.*

Índice general

Índice de figuras

Capítulo 1

Introducción

En los últimos años, se ha producido un enorme crecimiento de Internet y del uso y aplicaciones de las redes de computadores. Se mueven ingentes cantidades de datos a través de Internet y de las redes privadas de empresas e instituciones. Esto hace que el control y la configuración de los dispositivos de red sean cada vez más complejos.

Las primeras redes comerciales se valían del protocolo Arcnet, utilizaban cable coaxial y empleaba conexiones de 2.5 Mbps, que en ese tiempo se consideraba alta velocidad

Arpanet fue creada durante la Guerra Fría, y su objetivo principal era que la información militar de los Estados Unidos no estuviera centralizada y pudiera estar disponible desde cualquier punto del país ante un eventual ataque ruso.

Junto con la primera red WAN en 1970 se creó el protocolo TCP/IP, que se sigue utilizando hasta hoy, siendo un estándar dentro de las redes informáticas. El protocolo TCP/IP, cuya característica principal es poder compartir información entre redes muy distintas entre sí, es realmente Internet.

En 1983, Paul Mockapetris y Jon Postel crearon el sistema de nombres de dominio (DNS) y las denominaciones .com, .org, y .gov, tan características de lo que hoy llamamos Internet.

La última etapa en el desarrollo de Internet fue la creación de la World Wide Web, a cargo de Tim Berners-Lee, quien a principio de los 90 inventó el sistema de links, fundamental para el crecimiento de la red de redes[1].

El diseño y gestión de redes se ha vuelto más innovador en los últimos años con la ayuda de SDN (redes definidas por software). Esta tecnología es el resultado de largo tiempo tratando de hacer las redes informáticas más programables.

Las redes de ordenadores son complejas y difíciles de manejar. Implican muchos tipos de equipos, desde enrutadores y conmutadores hasta cortafuegos o traductores de direcciones de red. Por lo general, los enrutadores y conmutadores ejecutan software de control complejo y distribuido que suele ser cerrado y propietario.

Tradicionalmente los administradores de red configuran los dispositivos de red individuales utilizando interfaces de configuración que varían entre los vendedo-

res e incluso entre diferentes productos del mismo fabricante. Aunque algunas herramientas de gestión de red ofrecen un punto de vista central para configurar la red, estos sistemas siguen funcionando a nivel de protocolos individuales, mecanismos e interfaces de configuración. Este modo de operación ha ralentizado la innovación, ha incrementado la complejidad, y ha inflado tanto el capital como los costes operativos de gestión de una red.

El paradigma de Redes Definidas por software (SDN, en inglés) está cambiando la forma en que se diseñan y gestionan las redes. Tiene dos características definitorias. En primer lugar, SDN separa el plano de control (que decide cómo manejar el tráfico) del plano de datos (que reenvía el tráfico de acuerdo con las decisiones que toma el plano de control). En segundo lugar, SDN consolida el plano de control, de manera que un único programa de software controla múltiples elementos del plano de datos. El plano de control SDN ejerce control directo sobre el estado en los elementos del plano de datos de la red (es decir, enrutadores, conmutadores y otras middleboxes) a través de una API bien definida.

Un conmutador OpenFlow tiene una o más tablas de reglas de administración de paquetes. Cada regla corresponde a un subconjunto de tráfico y realiza ciertas acciones en el tráfico que coincide con una regla. Dependiendo de las reglas instaladas por una aplicación de controlador, un conmutador OpenFlow puede comportarse como un enrutador, conmutador, cortafuegos, traductor de direcciones de red o algo intermedio.

Aunque el boom de SDN se ha hecho más palpable recientemente, muchas de las ideas relacionadas con esta tecnología han evolucionado en los últimos 20 años. De alguna manera, SDN revisa ideas de las redes de telefonía primigenias, que separan claramente los planos de control y de datos para simplificar la gestión de la red y el despliegue de nuevos servicios.

La historia se divide en tres etapas, cada una con sus propias contribuciones:

- Redes activas desde mediados de los años 90 hasta principios de los 2000, que introdujeron funciones programables en la red, llevando a una mayor innovación.
- Separación de datos de control y de datos (alrededor de 2001 a 2007), que desarrolló interfaces abiertas entre los planos de control y de datos.
- OpenFlow y los sistemas operativos de red (desde 2007 hasta alrededor de 2010), lo que representó la primera adopción generalizada de una interfaz abierta y desarrolló formas de escalabilidad y práctica de la separación de control y datos.

La virtualización de la red desempeñó un papel importante a lo largo de la evolución histórica de SDN tomando raíz como uno de los primeros casos de uso significativo para SDN.

A mediados de la primera década del siglo XXI, en la Universidad de Stanford, un grupo de investigadores creó el Clean Slate Program, enfocado en la experimentación en redes universitarias más tratables y locales, que dio lugar al protocolo OpenFlow.

Gracias a la adopción de OpenFlow en las empresas, que abrieron sus API para permitir a los programadores controlar ciertos comportamientos de reenvío, la versión inicial de este protocolo se estableció en los switches a través de una simple actualización de firmware, sin necesidad de actualizar el hardware.

OpenFlow, aunque utilice muchos de los principios de anteriores trabajos en la separación de planos, también aporta bastantes contribuciones como la generalización de dispositivos de red y funciones, la visión de un sistema operativo de red y técnicas de gestión distribuida del estado de dispositivos de red entre otras [8].

A día de hoy muchos son los que hablan de SDN no sólo como una evolución de la red sino como una auténtica revolución tecnológica. SDN no sólo representa un cambio en la tecnología que se aplica, sino también una nueva manera de pensar en cómo se aplica, comparable con lo que sucedió hace unos años en el mundo del datacenter con la virtualización de servidores y almacenamiento.

Esta característica abre un gran número de interesantes beneficios entre los que destacan [4]:

1. Redes abiertas basadas en estándares, lo cual implica una interoperabilidad con todos los fabricantes permitiendo así una integración tanto horizontal de las diferentes capas de red como vertical de los elementos de cómputo y de almacenamiento con la red.

2. Redes controladas de una forma central, lo que supone ahorro de costes en la gestión y monitorización además de garantías de calidad extremo a extremo.

3. Redes programables, aportando como principal beneficio la posibilidad de ligar las redes a los servicios y a las aplicaciones.

Ámbitos de aplicación a destacar[5]:

- Virtualización: redes virtuales de datacenters, redes virtuales de campus, funciones de red como un servicio.

- Interconexiones dinámicas: interconexiones virtuales privadas/Cloud Bursting, optimización multicapa.

- Automatización de funciones de red y servicios.

1.1. OpenFlow

Aunque muchos fabricantes han desarrollado sus propias soluciones, OpenFlow es el primer interfaz estándar definido entre el plano de control y de datos de una arquitectura SDN. Se implementa en ambos lados, en los dispositivos de infraestructura de red y el software de control de SDN. Podría compararse al conjunto de instrucciones de una CPU. El protocolo especifica primitivas básicas que pueden ser usadas por una aplicación software externa para programar el plano de datos de dispositivos de red. Para ello, se basa en el concepto de flujos para identificar el tráfico de red basado en reglas de coincidencia predefinidas que se pueden programar de forma estática o dinámica por el software de control de SDN.

OpenFlow permite definir como debe fluir el tráfico a través de dispositivos de red basados en parámetros tales como patrones de uso, aplicaciones y recursos de la nube. Dado que OpenFlow permite que la red sea programada por flujo, una arquitectura basada en OpenFlow proporciona un control extremadamente granular permitiendo a la red responder a los cambios en tiempo real en los niveles de aplicación, usuario y sesión. El enrutamiento actual basado en IP no proporciona este nivel de control, ya que todos los flujos entre dos puntos finales deben seguir la misma ruta a través de de la red, independientemente de sus diferentes requisitos.

Aunque OpenFlow inicialmente se aplicaba a redes basadas en Ethernet, la conmutación OpenFlow puede extenderse a un conjunto mucho más amplio de casos de uso. Los dispositivos de red pueden soportar el reenvío basado en OpenFlow, así como el reenvío tradicional, lo que hace que sea muy fácil para las empresas y los operadores introducir progresivamente las tecnologías SDN basadas en OpenFlow, incluso en entornos de red de varios proveedores. Así, OpenFlow está siendo ampliamente adoptado por los proveedores de infraestructura, que normalmente lo han implementado a través de un simple firmware o actualización de software.

Tablas de flujo OpenFlow

El pipeline de un switch OpenFlow contiene múltiples tablas de flujo, cada una de las cuales puede contener múltiples entradas. El pipeline define como interactúan los paquetes con las tablas de flujo. Un switch OpenFlow debe tener al menos una tabla de flujo, y opcionalmente puede tener más de una. El procesamiento siempre comienza en la primera tabla de flujo y se utilizarán las tablas sucesivas dependiendo de la salida de la primera tabla. El proceso completo se muestra en la siguiente figura.

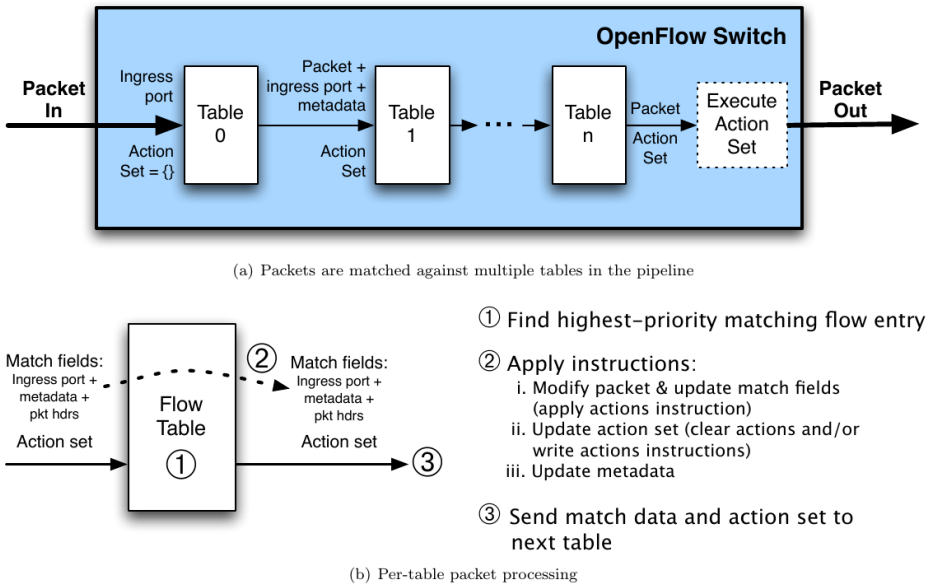


Figura 1.1: Flujo de paquetes a través del pipeline OpenFlow (Figura tomada de [9])

Cuando un paquete se procesa en una tabla de flujo, el paquete se compara con el patrón de cada una de las entradas de la tabla de flujo para seleccionar una entrada. Si en este proceso se encuentra una entrada compatible se ejecuta el conjunto de instrucciones asociado a esa entrada. Estas instrucciones pueden redirigir el paquete hacia otra tabla de flujo posterior donde se repite el mismo proceso. En la última tabla del pipeline, obviamente, no sería posible esta redirección. Cuando el conjunto de instrucciones no contiene una instrucción de redirección, el procesamiento se detiene en la tabla correspondiente y normalmente el paquete se reenvía a alguno de los puertos de salida del switch.

Entradas de la tabla de flujo

Cada entrada de la tabla de flujo consiste en:

- Campos patrón (match fields): Los patrones estarán compuestos por el puerto de ingreso, cabeceras de paquete y opcionalmente metadatos procedentes de una tabla anterior.
- Prioridad: Orden de precedencia de la entrada.
- Contadores: Se actualizan cada vez que un paquete casa con el patrón.
- Instrucciones: Permiten modificar el conjunto de acciones o el procesamiento en pipeline.
- Timeouts: Tiempo máximo o tiempo en desuso antes de que elimine la entrada de la tabla de flujo.

- Cookie: Es un valor opaco elegido por el controlador. No se usa para procesar los paquetes.

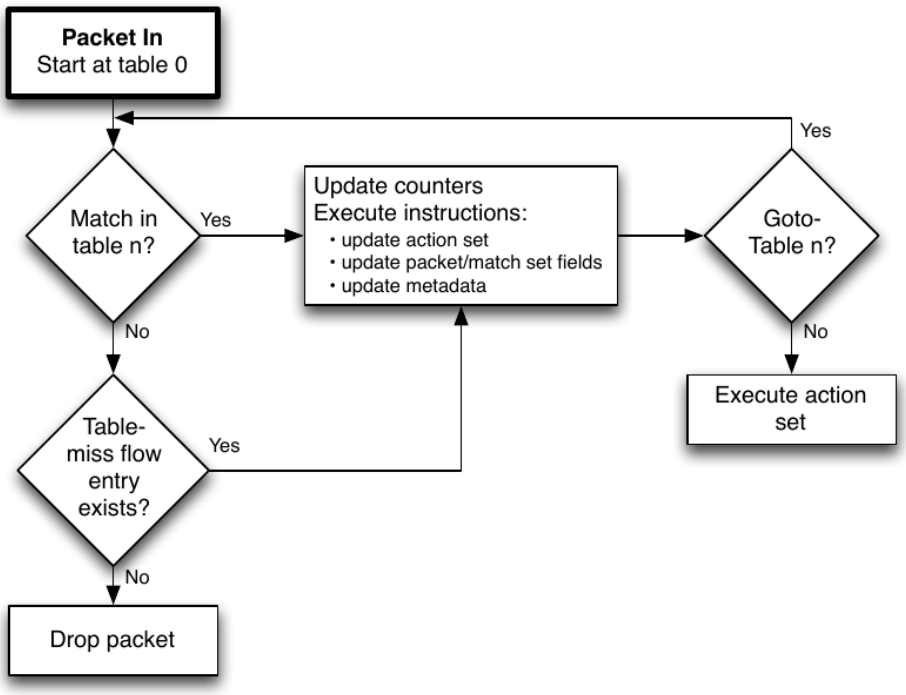


Figura 1.2: Diagrama de flujo del proceso de matching (Figura tomada de [9])

Cuando se recibe un paquete, un switch OpenFlow lleva a cabo las funciones que se muestran en el diagrama de flujo de la figura anterior. El switch comienza haciendo una búsqueda en la primera tabla de flujo en base a los match fields. Los match fields dependen del tipo de paquete y pueden ser, por ejemplo, las direcciones MAC, las direcciones IP. Además se pueden tener en cuenta los metadatos y el puerto de ingreso. Un paquete casa con una entrada de la tabla de flujo cuando los match fields coinciden con los valores especificados en esta. Si el valor de la entrada es ANY el campo se omite. El switch también puede soportar máscaras de bits que permiten buscar coincidencias parciales. Una vez que se encuentra una entrada coincidente, se incrementan los contadores y se aplica el conjunto de instrucciones [6].

Beneficios de SDN basadas en OpenFlow

Para las empresas SDN hace que la red sea un diferenciador competitivo, no sólo un coste inevitable.

OpenFlow permite hacer frente a grandes anchos de banda, a la naturaleza cambiante de las aplicaciones de hoy en día y reducir significativamente las operaciones y la complejidad de la administración.

Los beneficios que las empresas pueden lograr a través de una arquitectura basada en SDN OpenFlow incluyen [10]:

- Control centralizado de entornos de varios proveedores.
- Reducción de la complejidad a través de la automatización.
- Mayor tasa de innovación.
- Mayor fiabilidad y seguridad de la red.
- Control de red más granular.
- Mejor experiencia de usuario.

1.2. Objetivos

El objetivo del TFG, es acercarnos a la tecnología SDN, conocer el protocolo OpenFlow y desarrollar un caso de uso en redes corporativas basado en este paradigma. El caso de uso será el control del tráfico de red en base a distintos grupos de usuarios.

La red de comunicaciones se estructurará como una red plana, sin subredes, pero en la que se distinguen distintos grupos de usuarios que podrán conectarse entre sí o no según una serie de políticas. Para ello, antes de poder transferir datos los usuarios deberán autenticarse con la red. Cuando se autentiquen los usuarios, se comprobará a que grupo pertenecen y se aplicarán los privilegios de conectividad correspondientes.

Para cumplir con estos objetivos generales se deberán alcanzar los siguientes objetivos específicos:

- Diseñar una arquitectura de interconexión entre el conexionado de switches, el controlador de la red y el servidor de autenticación.
- Desarrollar un prototipo del servidor de autenticación y las bases de datos necesarias para guardar los privilegios de conectividad entre grupos de usuarios.
- Desarrollar el software de control necesario para que la red opere según las reglas establecidas.

1.3. Herramientas

En esta sección se describen brevemente las herramientas utilizadas para elaborar el software de control, así como la simulación de una red de pruebas cuyo diseño se indicará más adelante.

1.3.1. Ryu

Ryu es una infraestructura basada en componentes para redes definidas por software. Los componentes proveen una API bien definida que permite a los desarrolladores crear nuevas aplicaciones de control. Soporta varios protocolos para manejar dispositivos, entre ellos OpenFlow. Actualmente soporta las versiones 1.0, 1.2, 1.3 y 1.4 de este estándar.

La mayoría de los controladores tienen las siguientes características:

- Capacidad para recibir y gestionar eventos.
- Capacidad de analizar los paquetes entrantes y de crear paquetes nuevos para enviarlos hacia la red.
- Capacidad para crear y enviar mensajes OpenFlow para reprogramar los conmutadores.

La forma de proceder de un programa de control normalmente es la siguiente:

1. Recibe un paquete de un conmutador que genera un evento de tipo PACKET IN.
2. La rutina de atención al evento procesa el paquete, decidiendo las acciones a tomar, que pueden pasar por crear un nuevo paquete y enviarlo, o crear una nueva entrada en la tabla de flujo, para que el paquete entrante y los sucesivos se reenvíen o se eliminen.
3. Enviar el paquete creado o enviar un mensaje de FLOW MOD para insertar un flujo.

```
1  class L2Forwarding(app_manager.RyuApp):
2  def __init__(self, *args, **kwargs):
3  super(L2Forwarding, self).__init__(*args, **kwargs)
4
5  #con este evento y el metodo packet in handler, permitimos
6  #a la aplicacion recibir paquetes enviados por el switch al controlador.
7  @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
8  def packet_in_handler(self, ev):
9    msg = ev.msg
10   in_port = msg.match['in_port']
11
12   pkt = packet.Packet(msg.data)
13   eth = pkt.get_protocol(ethernet.ethernet)
14   dst = eth.dst
15
16   match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
17   actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD)]
18   inst = [parser.OFPInstructionActions(
19   ofproto.OFPIT_APPLY_ACTIONS, actions)
20   ]
21   mod = parser.OFPFlowMod(datapath=datapath, priority=0,
22   match=match, instructions=inst)
23   datapath.send_msg(mod)
```

Figura 1.3: Ejemplo de aplicación Ryu.

Para crear una nueva entrada en la tabla de flujos se debe especificar el patrón, el conjunto de instrucciones y las acciones.

En la figura 1.3, se muestra el método `packet_in_handler` de una aplicación Ryu. En las líneas 13-15 se analiza el paquete de datos entrante y se obtiene la dirección de destino de dicho paquete. En la línea 17 creamos el patrón que coincide con el paquete enviado, en este caso nos bastará con indicar la dirección de destino, marcamos que se haga un *flood* en la línea 18 en el `actions` y en el `inst` hacemos que se apliquen las acciones. Finalmente, se crea el mensaje de modificación de flujo en la línea 22 con el match, las inst la prioridad y se manda este mensaje al controlador para que se aplique esta regla, al paquete y a los sucesivos que casen con el patrón (*match*).

1.3.2. Mininet

Mininet es un emulador de red que crea una red de máquinas virtuales, switches, controladores, y enlaces. Los hosts Mininet ejecutan software de red estándar de Linux, y sus switches son compatibles con OpenFlow para enrutamiento personalizado altamente flexible y redes definidas por software.

Mininet apoya la investigación, el desarrollo, el aprendizaje, la creación de prototipos, pruebas, depuración, y cualquier otra tarea que pudiera beneficiarse de tener una red experimental completa en un ordenador portátil u otro PC.

Las características de Mininet son las siguientes:

- Proporciona un sencillo y barato banco de pruebas de red para el desa-

rrollo de aplicaciones OpenFlow.

- Permite a múltiples desarrolladores trabajar de forma simultánea e independiente en la misma topología.
- Soporta pruebas de regresión a nivel de sistema, que son repetibles y fácilmente empaquetadas.
- Permite realizar pruebas en topologías complejas, sin la necesidad de cablear una red física.
- Incluye un CLI que es consciente de la topología y OpenFlow-conscientes, para las pruebas de toda la red de depuración o de testeo.
- Soporta topologías personalizadas, e incluye un conjunto básico de topologías parametrizadas.
- Proporciona una API de Python sencilla y extensible para la creación de redes y la experimentación.

Las redes Mininet ejecutan código real, incluyendo las aplicaciones estándar de red Unix/Linux, así como el verdadero núcleo y la pila de red de Linux (incluyendo cualquier extensión del kernel disponible, siempre y cuando sean compatibles con los espacios de nombres de la red.)

Debido a esto, el código a desarrollar y probar en Mininet, por un controlador OpenFlow, switch modificado, o anfitrión, pueden pasar a un sistema real con cambios mínimos. Es importante destacar que esto significa que un diseño que funciona en Mininet por lo general puede pasar directamente a los switches físicos. [7]

Cabe mencionar que se pueden crear diferentes topologías de red para probar que disposición puede ser de más utilidad para una posible implementación o más óptima para optimizar los recursos que se tengan o para depurar el código de una aplicación de control fuera del entorno de producción. En la figura 1.4, se muestra un código de ejemplo para crear una topología. En la sección 2.2 se explica más detalladamente este código.

```
1
2  switch = net.addSwitch('s1', mac='00:00:00:00:00:01')
3
4  net = Mininet( topo=None, build=False )
5
6  hosts = []
7  for n in range(1,5):
8    hosts.append( net.addHost('h%d' % (n), ip = '0.0.0.0',
9    mac = '00:00:00:00:%02d:00' % n))
10   net.addLink(hosts[n-1],switch)
```

Figura 1.4: Topologías Mininet.

1.3.3. VirtualBox

VirtualBox es un software de virtualización de código abierto proporcionado por Oracle adecuado tanto para la empresa como para el hogar. Paquetes de virtualización como VirtualBox permite a un usuario ejecutar una instancia secundaria de otro sistema operativo con fines de compatibilidad o de prueba [11].

1.3.4. Python

Python es un lenguaje de programación de alto nivel interpretado, orientado a objetos con semántica dinámica. Sus estructuras de datos de alto nivel, combinadas con el mecanografiado dinámico y el enlace dinámico, lo hacen muy atractivo para el desarrollo rápido de aplicaciones, así como para usarlo como un lenguaje de script o para conectar componentes existentes. La sintaxis sencilla y fácil de aprender de Python hace hincapié en la legibilidad y por lo tanto reduce el coste del mantenimiento del programa. Python soporta módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización de código. El intérprete de Python y la amplia biblioteca estándar están disponibles en forma de código fuente o binario sin cargo para todas las plataformas principales, y se pueden distribuir libremente [12].

1.3.5. Sqlite

SQLite es una librería que implementa un motor de base de datos SQL transaccional autónomo, sin servidor y sin configuración. El código para SQLite es de dominio público y por lo tanto es de uso libre para cualquier propósito, comercial o privado. SQLite es la base de datos de mayor despliegue en el mundo, incluyendo varios proyectos de alto perfil. SQLite es un motor de base de datos SQL embebido. A diferencia de la mayoría de las otras bases de datos SQL, SQLite no tiene un proceso de servidor independiente. SQLite lee y escribe directamente en discos normales [15].

1.3.6. Cygwin

Cygwin sirve para simular una consola de comandos de Linux en un entorno Microsoft Windows, se ha utilizado para facilitar el acceso a la máquina virtual mininet vía ssh; de esta manera se pueden abrir xterms de los terminales de la topología haciendo uso también de Xming server.

1.3.7. CherryPy

CherryPy es un entorno de desarrollo HTTP orientado a objetos estilo Python [16]. Permite crear aplicaciones web complejas, con poco o sin ningún conocimiento de los protocolos internos. CherryPy permite que los desarrolladores construyan aplicaciones web en la misma manera que construirían cualquier otro programa orientado a objetos en Python. Esto normalmente resulta en menos código fuente desarrollado en menos tiempo.

Las aplicaciones CherryPy son por lo general muy simples. Funciona sin hacer cambios, el comportamiento predeterminado es lo suficientemente sensible para que se use sin mucha configuración o personalización. El servidor web incluido permite que uno tenga aplicaciones web en cualquier parte donde esté instalado Python.

Capítulo 2

Descripción del sistema

2.1. Descripción general y conexionado de red

Lo que se plantea en este trabajo de fin de grado es trabajar con una red plana por lo que a priori podría haber conexión de todos con todos, cosa que en la práctica sería inviable, por lo que se seguiría alguna estrategia para segmentar la red, como el uso de VLANs. Sin embargo, mediante el uso de SDN, la segmentación de la red se realizará en base a usuarios, que deberán identificarse con la red para poder transmitir datos. En la figura 2.1, se muestra la topología simplificada de la red que se va a utilizar. Se pueden distinguir tres elementos básicos: Switch o conexionado de switches, el controlador y el autenticador. Los dos primeros son elementos básicos de cualquier infraestructura basada en SDN. Los switches se encargarán del reenvío de paquetes en base a las reglas instaladas por parte del controlador, utilizando el protocolo OpenFlow. El otro elemento básico de la red es el autenticador, que es un servicio encargado de identificar a los usuarios y de actualizar los permisos de estos en el controlador, para que éste aplique las reglas necesarias en los switches. Cualquier conexión que se haga desde un host no autenticado a cualquier destino que no sea el autenticador irá directamente al controlador donde presumiblemente se descartará el paquete. Los hosts autenticados podrán comunicarse con otros hosts del mismo grupo de usuarios, creándose de esta manera una segmentación de la red plana.

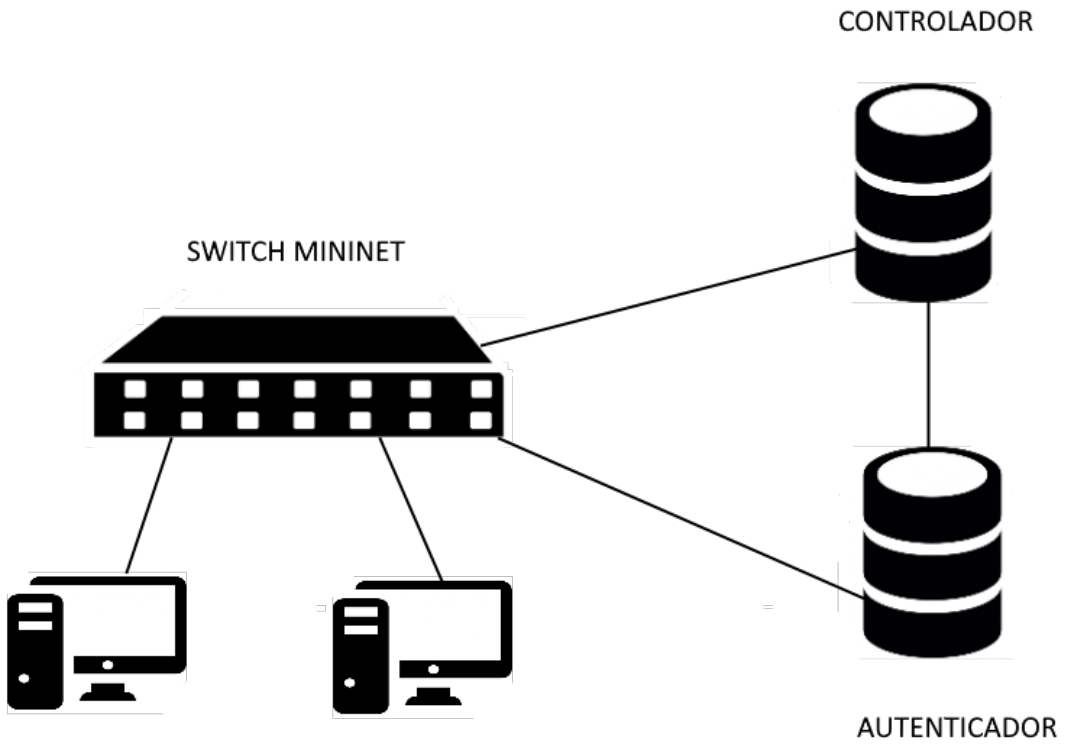


Figura 2.1: Topología

2.2. Descripción del entorno de simulación empleado

Para el desarrollo del supuesto se han utilizado 3 máquinas virtuales Debian GNU/Linux donde se ha instalado OpenFlow, Mininet y Ryu. La primera ha sido destinada a ser el controlador donde se ejecutarán las aplicaciones de control de la red. La segunda es el autenticador web, los usuarios al logearse se autenticarán contra este. La tercera se encargará de crear la topología mediante de Mininet, en concreto gracias a un script con el que conseguimos que la interfaz eth1 del switch se corresponda con la interfaz eth1 de la máquina virtual. Las conexiones entre las distintas máquinas así como las direcciones de red que les corresponde a cada una se especifican en la figura 2.2

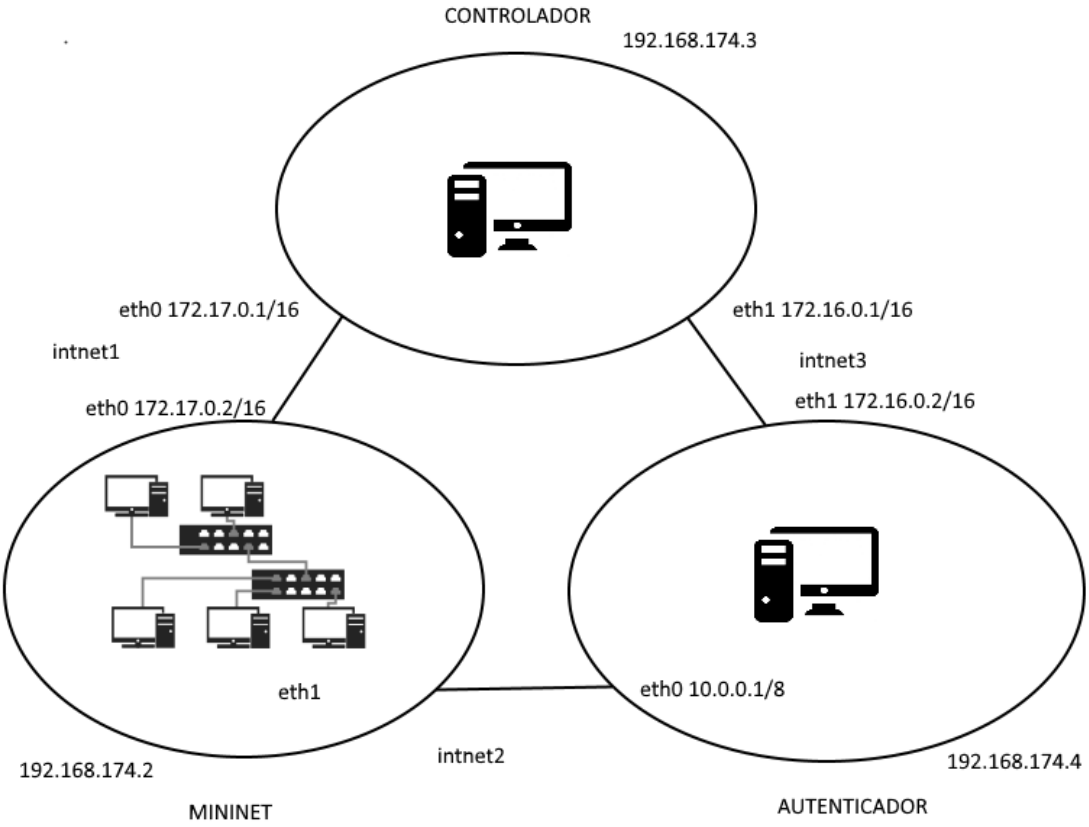


Figura 2.2: Topología virtual box

Para la conexión entre máquinas se configura en Virtualbox que las interfaces de interconexión sean red interna y se les asigna la dirección correspondiente.

La conexión entre el autenticador y Mininet es para la autenticación de los usuarios, la conexión entre Mininet y el controlador para aplicar las políticas desde el controlador a el switch Openflow además del uso del DHCP y el resto de códigos, y la conexión entre el autenticador y el controlador es para enviar las políticas de los usuarios desde el autenticador a el controlador.

Al estar los hosts de Mininet en la misma red que la interfaz eth0 del autenticador, éstos se pueden conectar a al mismo para autenticarse.

```
1  if __name__ == '__main__':
2  setLogLevel( 'info' )
3
4  intfName1 = 'eth1'
5  checkIntf( intfName1 )
6  info( '*** Creating network\n' )
7  net = Mininet( topo=None, build=False )
8
9  switch = net.addSwitch('s1', mac='00:00:00:00:00:01')
10 controller = net.addController('c0', controller = RemoteController,
11 ip='172.17.0.1')
12
13 info( '*** Adding hardware interface', intfName1, 'to switch',
14 switch.name, '\n' )
15
16 _intf1 = Intf( intfName1, node=switch )
17
18 hosts = []
19 for n in range(1,5):
20   hosts.append( net.addHost('h%d' % (n), ip = '0.0.0.0',
21   mac = '00:00:00:00:%02d:00' % n))
22   net.addLink(hosts[n-1],switch)
23
24 net.build()
25 net.start()
26
27
28 for n in range(0,4):
29   hosts[n].cmdPrint('dhclient -v ' + hosts[n].defaultIntf().name)
30
31 CLI( net )
32 net.stop()
```

Figura 2.3: Código para crear una topología en estrella en Mininet.

La figura 2.3 muestra el código para crear la topología mininet, se comprueba que exista la interfaz (línea 5), en este caso eth1, se añade un switch con su mac (línea 9), se le indica que el controlador es remoto y cual es su dirección IP (línea 9-11), se añade la interfaz eth1 al switch y se añaden los host con dirección ip 0.0.0.0 temporal y dirección mac única (línea 19-22). Una vez creada la topología se hace un dhclient en todos los hosts para tener una dirección IP asignada mediante DHCP (línea 28-29). En este primer caso, sólo se utiliza un switch y con 5 hosts conectados, siendo una topología en estrella.

```
1  controller = net.addController('c0', controller = RemoteController, ip='172.17.0.1')
2
3  switch_prev = net.addSwitch('s1', dpid = '00:00:00:00:00:1')
4
5  for m in range(2,5):
6
7    switch = net.addSwitch('s%d' % (m), dpid = '00:00:00:00:00:%02x' % (m))
8    net.addLink(switch_prev, switch)
9
10   for n in range(1,5):
11     hosts.append( net.addHost('h%d' % (num), ip = '0.0.0.0',
12                    mac = '00:00:00:00:%02x:00' % num))
13     net.addLink(hosts[num-1],switch)
14     num = num + 1
```

Figura 2.4: Código para crear una topología en árbol en Mininet.

En esta figura 2.4 se crea una topología más compleja en la que se utilizan 3 switches y en este caso se trata de una topología en árbol. Se crean los switches mediante un bucle y se añaden los hosts(línea 5-14), a diferencia de la topología anterior ahí que crear los enlaces entre los switches (línea 8).

Capítulo 3

Software de control y autenticación

El software de control deberá gestionar y aplicar las políticas de conectividad entre grupos y de seguridad, de modo que en una red plana a nivel de enlace, puedan separarse distintos grupos en base a esas políticas. Para ello se hará uso del controlador Ryu y del protocolo OpenFlow. La topología se simulará mediante Mininet, tal y como se ha comentado en el capítulo 2.

En la figura 3.1, se muestran los distintos módulos que componen el sistema y la relación entre ellos.

Para poder separar una red plana en distintos segmentos en función de políticas de conectividad es necesario disponer primero de un módulo para almacenar y gestionar estas políticas, que se implementará en un servidor separado – el servidor de autenticación. Una vez se han establecido distintas políticas en base a usuarios y/o grupos, éstos deberán ser capaces de autenticarse contra el sistema y que éste les permite llevar a cabo aquellas conexiones autorizadas. El proceso de autenticación se realizará mediante un servicio web instalado en el servidor de autenticación.

Por otra parte, cuando un usuario se autentica, se deberán implantar las políticas correspondientes en la red. Las políticas se harán efectivas mediante la inserción de reglas en las tablas de flujo de los switches OpenFlow que componen la topología de red. La inserción de reglas se lleva a cabo desde una aplicación controladora, que se ha implementado mediante el controlador Ryu, descrito en la sección 1.3.1. Ryu permite dividir las funcionalidades en distintas aplicaciones Ryu, que a priori funcionan de manera independiente, pero que pueden comunicarse mediante mecanismos establecidos para ello si es necesario. El control de esta red se ha dividido en cuatro aplicaciones: Servicio de DHCP, Servicio de ARP, Aplicación de políticas y switching.

Hay que tener en cuenta que, al tratarse de una red plana a nivel de enlace, cuando la red alcanza un cierto tamaño, debe limitarse el tráfico broadcast, ya que este afecta a todos los nodos de la red. Existen dos protocolos esenciales en

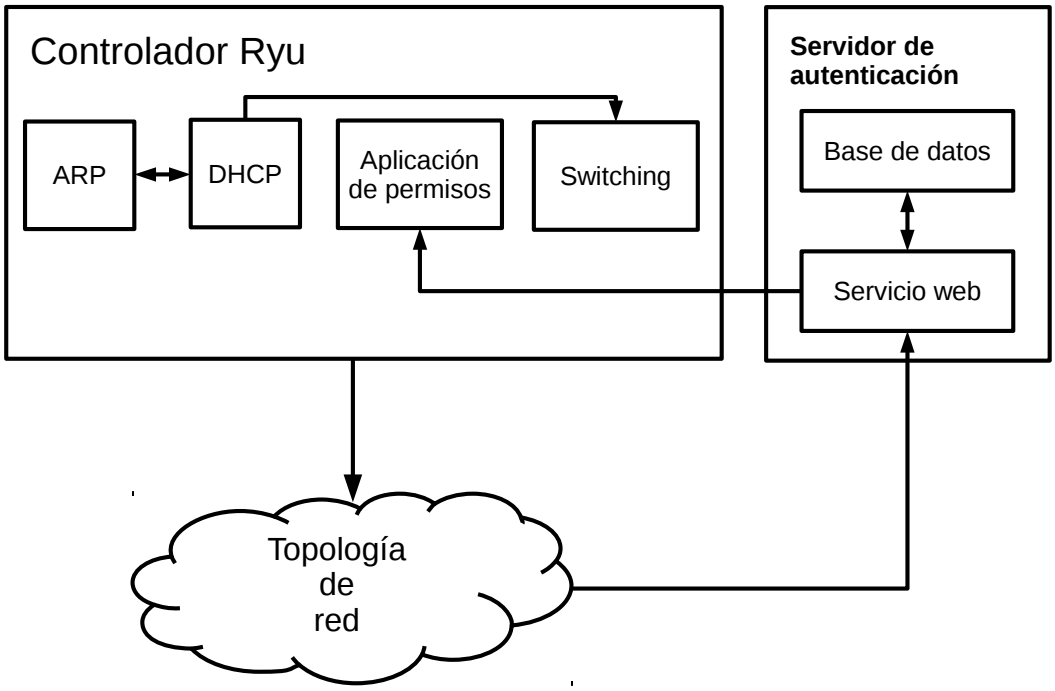


Figura 3.1: Módulos del sistema y su relación.

las redes TCP/IP que se basan en el envío de paquetes broadcast. El primero de ellos es el protocolo ARP, que se utiliza para obtener la dirección MAC correspondiente a una determinada dirección IP y el segundo es el protocolo DHCP, que permite a los host autoconfigurar su dirección IP. Ambos tipos de tráfico, y posiblemente alguno más, deberá evitarse. Por ello el software de control deberá gestionar las peticiones y respuestas ARP y DHCP, evitando que todos los nodos se vean afectados por el tráfico broadcast. Además, el servicio de DHCP permitirá asignar direcciones IP a los hosts y que estas sean conocidas en todo momento por el controlador.

Cuando un host se inicia por primera vez deberá obtener su dirección mediante el protocolo DHCP y a continuación, como se ha comentado anteriormente, deberá proceder a conectarse con el autenticador para que la red le otorgue sus permisos. La aplicación directa de estos permisos es responsabilidad de otra aplicación Ryu, que es la que recibe los permisos del servidor de autenticación, en el momento del login, y a continuación inserta las reglas de conexionado pertinentes en los switches. Por otro lado, en el momento del logout se deberán desinstalar las reglas de los switches.

Finalmente, la última aplicación, se encargará del switching propiamente dicho, es decir, de conmutar aquellos paquetes permitidos a través de la red.

En las siguientes secciones se describirán con detalle cada uno de los módulos mencionados anteriormente.

3.1. Servidor de autenticación

Lo primero que tendrán que hacer los usuarios de los terminales dentro de la red de Mininet será introducir sus credenciales para comprobar en el servidor de autenticación si este usuario existe o no y en caso positivo traer las reglas que se le deban aplicar para poder tener conexión dentro de la red.

Por defecto todos los usuarios de la red tienen permitida la conexión con el servidor de autenticación (10.0.0.1) para que puedan autenticarse. Hay que recordar que este servicio a priori puede estar instalado en una máquina distinta al resto del sistema de control. Se compone de una base de datos que almacena las políticas y un servicio web que permite a los usuarios autenticarse.

Las políticas de red que se van a aplicar a la ya mencionada red plana tanto a nivel de usuario o a nivel de grupo, para poder aplicarlas es necesario conocer su dirección IP, el protocolo que atañe, la acción a realizar y optativamente los puertos y la prioridad de la política.

3.1.1. Base de datos

Para guardar tanto los usuarios, los grupos y las diferentes políticas a aplicar se ha optado por montar una base de datos de SQLite.

Las tablas de la base de datos son:

- Users (user, pass, uid, gid)
- Groups (name, gid)
- Policies (policy, protocol, o port, d port, action, priority)
- Connections (gid1, gid2, policy)

La tabla Users contiene el nombre de usuario, su contraseña, el id de usuario que le corresponde y el id del grupo al que pertenece.

La tabla Groups contiene los distintos grupos que existen con su id.

En la tabla Policies se guarda cómo son las políticas aplicables mediante un id, el protocolo usado(udp, tcp,...), el puerto de origen de la comunicación, el puerto de destino, si ese paquete se acepta o no y la prioridad que debe tener esa regla.

Las tablas GConnections y UConnections se utilizan para indicar entre que dos grupos o usuarios se aplican las políticas indicadas en la tabla Policies.

3.1.2. Servicio web

Para el desarrollo del servidor se ha hecho uso de CherryPyWSGIServer de la librería web y sqlite3 para la base de datos, la conexión con el controlador se muestra en 3.3.

La función que cumple el autenticador es recibir las credenciales que introduzca el usuario y si son correctas comunicar al controlador que dicho usuario está autenticado. Con este propósito se ha desarrollado servicio web, cuya interfaz se muestra en la figura 3.2. Dicha interfaz se ha desarrollado en lenguaje Python, mediante el paquete CherryPyWSGIServer explicado en la sección 1.3.7.

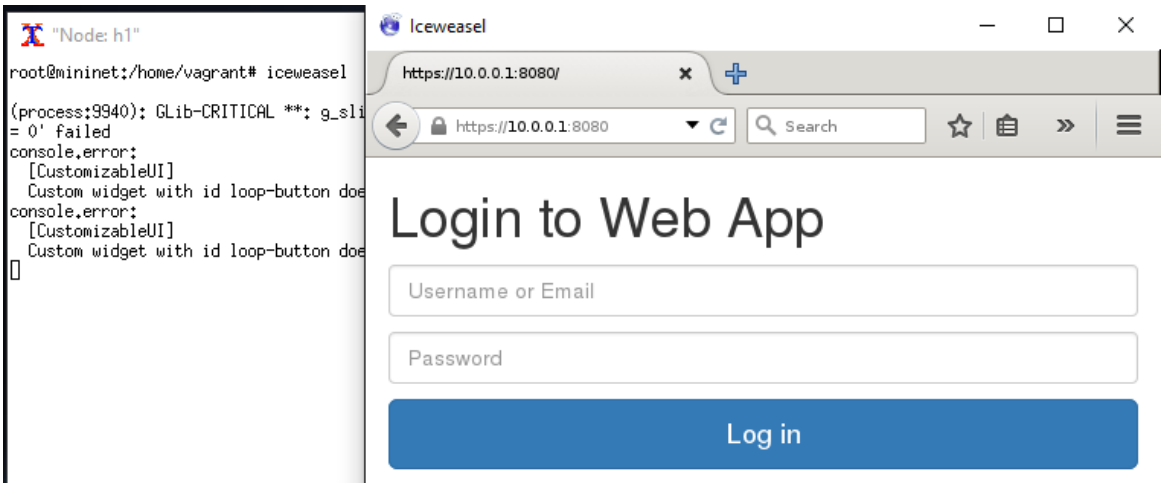


Figura 3.2: Autenticador

La autenticación se realiza mediante el fragmento de código que se muestra en la figura 3.3.

```
1  class Login:
2    def POST(self):
3      post_data=web.input(login="", password="")
4      for a in allowed:
5        if ([post_data['login'], post_data['password']] == a):
6          logged = post_data['login']
7          passwd = post_data['password']
8          session.logged_in = True
9          conn = sqlite3.connect("SDN.db")
10         cursor = conn.cursor()
11         cursor.execute('SELECT uid,gid FROM USERS WHERE user="'+logged+'"'
12                        'AND password ="'+passwd+'"')
13
14         for row in cursor:
15           uid = row[0]
16           gid = row[1]
17         uid=int(uid)
18         gid=int(gid)
19
20
21         session.uid = uid
22         session.gid = gid
23
24         j = self.get_policies(uid, gid, web.ctx['ip'])
25
26         sock = socket.socket(socket.AF_INET, # Internet
27                              socket.SOCK_STREAM)
28         sock.connect(('172.16.0.1', 6666))
29         js = json.dumps(j)
30
31         sock.send(struct.pack('I',len(js)))
32         sock.send(js)
33         sock.close()
34         return '<h1>You are logged in</h1><a href="/logout">Logout</a>'
35
36       session.logged_in = False
37       MESSAGE = "LOGOUT"
38       sock = socket.socket(socket.AF_INET, # Internet
39       socket.SOCK_DGRAM) # UDP
40       sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))
41       return '<h1>Inicio de sesion fallido!!! incorrecta.</h1>'
42              '<a href="/">Volver a intentar</a>'
```

Figura 3.3: Código de la clase Login.

Primero, se comprueba si la contraseña y usuario introducidos están entre los registrados en base de datos (línea 5) y si es así se busca en base de datos (línea 11) el uid y gid de este usuario para poder recuperar las políticas que se deben aplicar a éste y se crea un socket hacia el controlador para enviárselas en un mensaje.

Tras la autenticación del usuario se tendrá un hash con el id de usuario, el id de grupo y un otros dos hash uno para las políticas de grupo y otro para las de usuario.

Foto del hash

Tal y como se muestra en el esquema general de módulos de la figura 3.1,

dichas políticas se enviarán al controlador Ryu para que éste las aplique sobre la red. La comunicación autenticador-controlador se va a realizar mediante un socket al puerto 6666 una vez el usuario se autentique. El autenticador le envía al controlador toda la información para poder aplicar las políticas que le correspondan a ese usuario.

3.2. Aplicaciones de control Ryu

En esta sección se describen las aplicaciones de control Ryu que se han desarrollado para controlar la red. Estas aplicaciones son: Servidor DHCP, Servidor ARP, Aplicación de permisos y switching.

3.2.1. Servidor DHCP

En nuestra red es muy importante controlar las direcciones IP de todos los terminales para poder gestionar el envío de paquetes entre distintos grupos y usuarios, por este motivo se ha implementado un servicio de DHCP que otorgue a cada ordenador una dirección única y no haya conflictos entre los distintos terminales. Además, dicho servicio se implementa en el controlador permitiendo así el control del tráfico broadcast.

Para entender este último aspecto, primero debe revisarse el funcionamiento normal de DHCP. DHCP significa Protocolo de configuración de host dinámico. Como su nombre indica, DHCP se utiliza para controlar la configuración de red de un host a través de un servidor remoto.

Los pasos por los que pasa un intento de configuración DHCP son:

1. Cuando el equipo cliente se inicia, se envía un mensaje DHCPDISCOVER hacia el servidor. Como no hay información de configuración de red en el cliente, el mensaje se envía con 0.0.0.0 como dirección de origen y 255.255.255.255 como dirección de destino. El cliente entra en la etapa de inicialización durante este paso.

2. Cuando el servidor DHCP recibe el mensaje de solicitud DHCPDISCOVER, responde con un mensaje DHCPOFFER. Este mensaje contiene todos los ajustes de configuración de red requeridos por el cliente. Por ejemplo, el campo yaddr del mensaje contendrá la dirección IP que se asignará al cliente. De forma similar, la información de la máscara de subred y de la pasarela se rellena en el campo de opciones. Además, el servidor rellena la dirección MAC del cliente en el campo chaddr. Este mensaje se envía como un mensaje de difusión (255.255.255.255) para que el cliente lo reciba directamente. El cliente entra en la etapa de selección durante este paso.

3. El cliente forma un mensaje DHCPREQUEST en respuesta al mensaje DHCPOFFER y lo envía al servidor indicando que quiere aceptar la configuración de red enviada en el mensaje DHCPOFFER. El mensaje DHCPREQUEST seguirá conteniendo la dirección de origen como 0.0.0.0, ya que el cliente todavía no puede usar la dirección IP que se le haya pasado a través del mensaje DHCPOFFER. El cliente entra en la fase de solicitud durante este paso.

4. Una vez que el servidor recibe DHCPREQUEST desde el cliente, envía el mensaje DHCPACK indicando que ahora el cliente puede utilizar la dirección IP asignada a él. El cliente entra en el estado enlazado durante este paso.

Como se ha indicado en el párrafo anterior, cada una de las etapas de la configuración mediante DHCP implica el intercambio de una serie de paquetes, cuyo flujo se muestra en la figura 3.4.

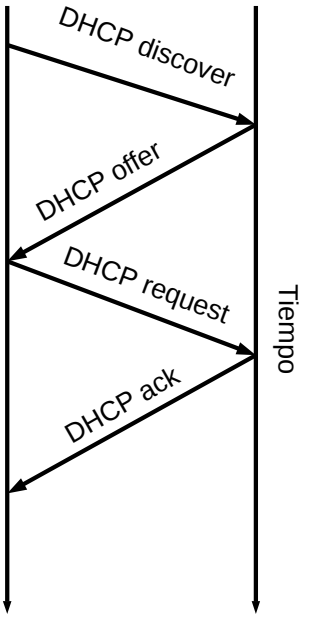


Figura 3.4: DHCP

La dirección IP asignada por el servidor DHCP al cliente DHCP está en un contrato de arrendamiento. Una vez finalizado el contrato de arrendamiento, el servidor DHCP puede asignar la misma dirección IP a cualquier otro host

o dispositivo que solicite la misma. Por lo tanto, el arrendamiento tiene que renovarse de vez en cuando. El cliente DHCP intenta renovar el contrato después de que haya transcurrido la mitad del tiempo de concesión. Esto se hace mediante el intercambio de mensajes DHCPREQUEST y DHCPACK. Mientras hace todo esto, el cliente entra en la etapa de renovación [3].

Como se ha visto anteriormente, las peticiones DHCP se hacen a través de broadcast, porque el PC no conoce la dirección del servidor DHCP en cuestión. Sin embargo, al tratarse de una red plana a nivel de enlace, el exceso de tráfico broadcast podría suponer un problema. Por ello, al ser una red SDN, el servicio de DHCP se implementa en el controlador, quien tiene una visión global de la red. Los host cliente realizan las peticiones exactamente como si estuvieran en una red tradicional, a la dirección de broadcast. En cambio, el swicth se programa para que filtre los paquetes DCHP y los envíe directamente al controlador, evitando así que lleguen al resto de hosts de la red. El controlador hace de servidor DHCP generando las respuestas correspondientes y el cliente no percibe el "engaño". Así la aplicación de servidor de DHCP se encarga de repartir direcciones IP a todos los hosts de la topología, el pool de direcciones son todas las disponibles en la red 10.0.0.0/16 menos la 10.0.0.1 reservada para el servidor de autenticación.

3.2.2. ARP

Al igual que sucede con el protocolo DHCP, el protocolo ARP está basado en el envío de paquetes broadcast a nivel de enlace que, para el caso de esta red, no son convenientes. Para evitar que las peticiones ARP lleguen a todos los hosts, se procede de forma similar a como se ha hecho en DHCP, es decir, se filtran los paquetes correspondientes a peticiones ARP y se envían al controlador. Éste tendrá que generar la respuesta para cada petición ARP indicando la dirección MAC correspondiente a la dirección IP peticionaria. La problemática de este enfoque reside en que el controlador deberá conocer las parejas de direcciones IP-MAC. Para ello, se aprovecha el hecho de que la asignación de IP se hace mediante DHCP, y en el momento de la asignación de IP, se guarda la pareja de direcciones en una tabla. Así en el momento de recibir una petición ARP, se recurre a dicha tabla y se genera la respuesta esperada por el host, nuevamente de forma transparente para éste. El mayor inconveniente que puede encontrarse a este mecanismo es que cuando un host no tiene IP asignada, no podrá comunicarse con los demás, pero como esto tampoco debería suceder, no es un problema.

En la figura 3.5 se muestra el fragmento de código que se encarga de generar la respuesta ARP en el controlador.

```
1  def handle_arp(self, datapath, port, pkt):
2    ether_pkt = pkt.get_protocol(ethernet.ethernet)
3    arp_pkt = pkt.get_protocol(arp.arp)
4
5    if (arp_pkt.dst_ip in self.ip_mac.keys() and
6      arp_pkt.opcode==arp.ARP_REQUEST):
7
8      ip = arp_pkt.dst_ip
9      mac = self.ip_mac[ip]
10
11
12     e = ethernet.ethernet(dst = ether_pkt.src, src = mac,
13     ethertype=ether.ETH_TYPE_ARP)
14
15     a = arp.arp(opcode = arp.ARP_REPLY, src_mac = mac, src_ip = ip,
16     dst_mac=arp_pkt.dst_mac, dst_ip=arp_pkt.src_ip)
17
18     p = packet.Packet()
19     p.add_protocol(e)
20     p.add_protocol(a)
21     self._send_packet(datapath, port, p)
```

Figura 3.5: Código de ARP.

Como se ve en la figura 3.5, en la quinta línea comprobamos que la dirección de destino del paquete ARP esté en el diccionario de direcciones ip-mac que se muestra en 3.2.5, en caso positivo, se creará un paquete ARP con la pareja direcciones IP-MAC de destino que se enviará al controlador para controlar los envíos de paquetes.

3.2.3. Aplicación de permisos

Antes de explicar la forma de hacer efectivas las políticas de conectividad entre grupos, se revisará el uso de las tablas de flujo del switch OpenFlow que intervienen en esta aplicación. Como se muestra en la figura 3.6, en esta aplicación se emplean cuatro tablas de flujo cuyo propósito se explica a continuación.

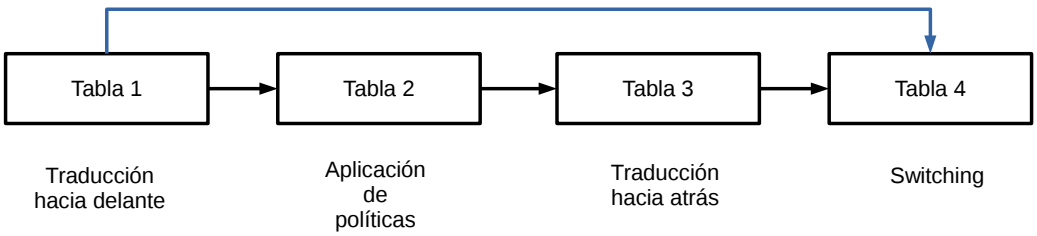


Figura 3.6: Uso de las tablas de flujo

Las tres primeras tablas son las responsables de implementar las políticas de conectividad, mientras que la cuarta tabla se utiliza para implementar el switching, que se explicará en la sección 3.2.4.

Las políticas de conectividad se han implementado en base a grupos de usuarios, identificados mediante un `gid`. Si dos usuarios pertenecen al mismo grupo pueden comunicarse entre sí, pero si dos usuarios pertenecen a dos grupos distintos, podrán comunicarse en base a unas reglas establecidas en base a los puertos de origen y destino y del protocolo, tal y como sucedería en un firewall.

Una opción para implementar esta funcionalidad es introducir un conjunto de reglas de conectividad para cada pareja de usuarios identificados que se comunican. Así el número de reglas aumentará dependiendo del número de usuarios comunicándose. Para evitar este crecimiento desmesurado, se aplicarán las políticas utilizando 3 tablas de flujo.

Supongamos que dos usuarios, el primero con dirección IP 10.0.0.5 y el segundo con dirección IP 10.0.0.8 van a realizar un intercambio de paquetes IP, y que el primer usuario pertenece al grupo 1 y el segundo al grupo 2. Lo que se hará es realizar una traducción de direcciones de la siguiente manera:

1. La IP de origen 10.0.0.5 se traducirá a 10.1.0.5 de modo que el gid queda registrado en la dirección IP. Lo mismo se hará con la dirección de destino, que pasa de 10.0.0.8 a 10.2.0.8. Esta es justamente, la misión de la primera tabla de flujo, realizar la traducción hacia delante. Para ello, el controlador deberá insertar las reglas correspondientes en la tabla 1 en el momento que este usuario envíe el primer paquete.

2. La segunda tabla se encarga de aplicar las políticas, basadas en grupos. Para que haya conectividad, el controlador deberá insertar una regla diciendo que los paquetes con origen en 10.1.0.0/16 y destino 10.2.0.0/16 deberán pasarse a la tabla 3, añadiendo al patrón las restricciones basadas en puertos si las hay. En caso de que la política sea de no conectividad, simplemente deberá descartar el paquete.

3. En la tabla 3, se produce una traducción hacia atrás, las direcciones de origen y destino se vuelven a dejar como estaban, es decir se pasa de 10.1.0.5 a 10.0.0.5 y de 10.2.0.8 a 10.0.0.8. Luego se pasa el paquete a la tabla 4.

Existe una excepción al procedimiento anterior, que hacen pasar los paquetes directamente desde la tabla 1 hacia la tabla 4, sin hacer la traducción (véase la flecha azul de la figura 3.6). Este es el caso de los paquetes que van dirigidos al servidor de autenticación, y que corresponde a los nodos no identificados, que únicamente deberán poder conectar con la dirección IP 10.0.0.1. Por otra parte, todos los paquetes que no casen con ninguna regla en la primera tabla de flujo deberán enviarse al controlador para que éste sea el que decida sobre el paquete e inserte las reglas correspondientes en las tablas de flujo.

3.2.4. Interconexionado de hosts (switching)

Este módulo se encarga de insertar las reglas correspondientes en la cuarta tabla de flujo, que se observa en la figura 3.6. Esté deberá hacer la función del switch de capa 2 o de un interconexionado de switches. Se ha implementado el switch único con propósitos de prueba y depuración, pero al ser este escenario poco realista se ha ampliado para poder operar un interconexionado de switches mediante SPF (*Shortest Path Forwarding*).

Ambos casos, son ejemplos de uso de Ryu que se pueden obtener de Internet [13, 2]. Sin embargo, éstos no son directamente aplicables. Hay que adaptar dos aspectos fundamentales. El primero de ellos es que las aplicaciones de ejemplo insertan las reglas en la tabla 1, pero aquí deben insertarse en la tabla 4. El segundo, está relacionado con el funcionamiento de un switch de capa 2 que se muestra en el diagrama de flujo de la figura 3.7. Concretamente, con las inundaciones que se realizan cuando el dispositivo no conoce el puerto por el que debe reenviar la trama (recuadro rojo).

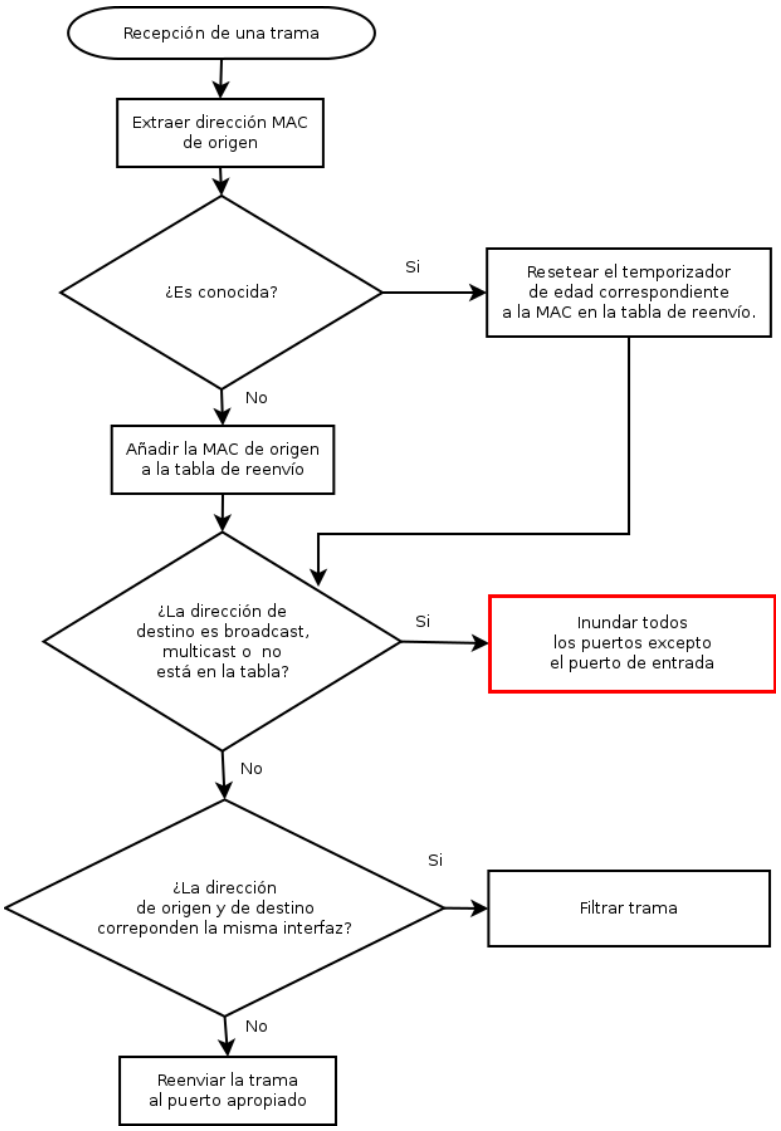


Figura 3.7: Diagrama de flujo del funcionamiento básico de un switch de capa 2.

Cuando el switch no conoce el puerto por el que debe enviar una trama, inunda todos los puertos excepto aquel por el que entró dicha trama. Eso es equivalente a lo que ocurre cuando un paquete va dirigido a la dirección de broadcast. Como se ha mencionado anteriormente, al tratarse de una red plana, el tráfico de tipo broadcast debe evitarse para evitar una reducción del rendimiento, además de por motivos de seguridad. Ambas aplicaciones de switching que se describirán en las secciones siguientes deben modificarse para que la tabla de reenvío no se obtenga por el mecanismo de autoaprendizaje, como en el caso de los switches convencionales. El aprendizaje de la tabla se hará en el momento en el que un host pide su dirección IP mediante DHCP, pues en este caso se conocerá la dirección MAC y el puerto. Así el switch únicamente podrá conmutar tramas entre hosts previamente configurados mediante DHCP y nunca será necesario realizar una inundación. En el código mostrado en la figura 3.5 se refleja este funcionamiento puesto que la única dirección de destino guardada en el diccionario ip_mac es la del autenticador cuando se manda un

paquete con cualquier dirección que no se encuentre en dicho diccionario no se tendrá en cuenta, solo cuando se intente autenticar o haya conexión entre hosts.

```
1 def assemble_ack(self, pkt, port):
2
3   self.ip_mac[str(ip)] = req_eth.src
```

Figura 3.8: Control ARP en el DHCP.

Como se ve en la figura 3.8, cuando se genera el paquete ACK se añade la dirección al diccionario IP-MAC.

Switch único

En este desarrollo se utiliza un solo switch por lo que todo el tráfico lo reenvía éste, si los paquetes que pasan a través están permitidos, se hará lo pertinente y si no se enviará el paquete al controlador donde se decide qué hacer.

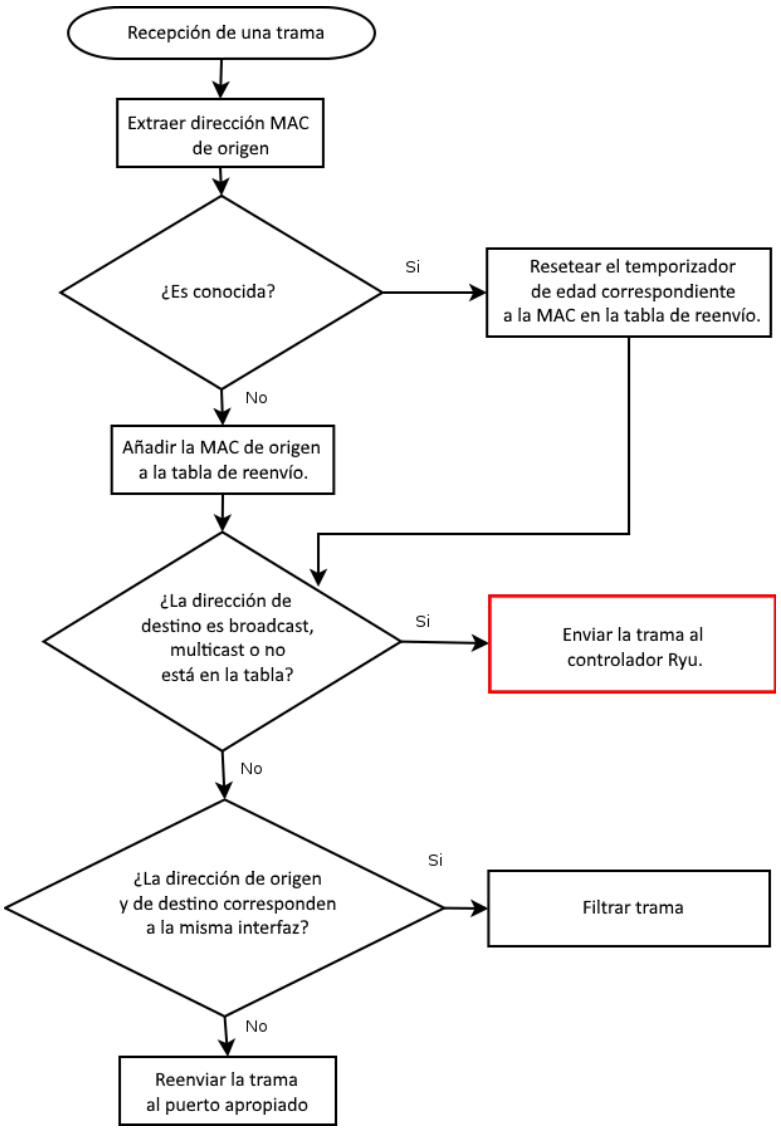


Figura 3.9: Diagrama de flujo del funcionamiento básico del switch implementado.

Shortest path forwarding (SPF)

Esta segunda implementación se basa en SPF cuyo objetivo es encontrar en cada salto cual es el siguiente paso a dar para llegar al nodo destino por el camino más corto, para ello:

Se calculará los caminos más cortos desde cada origen hasta cualquier otro host en la topología.

Para cada camino se instalarán reglas de flujo acordes en los switches. Se utilizan patrones que encajan con la direcciones de destino y las acciones que conducen al siguiente salto en el camino.

Se ha utilizado un algoritmo de caminos mínimos para calcular los costes entre enlaces, cabe destacar que no hay que tener en cuenta los posibles enlaces redundantes puesto que el controlador tiene el grafo completo de la red.

Esta implementación, como se comentó anteriormente, ha sido obtenida de

internet y se le han hecho modificaciones para que se adaptara a los requerimientos del desarrollo relacionados con las tablas de flujo a usar y el manejo adecuado de los casos en los que en un switch tradicional se requiere una inundación.

3.2.5. Miscelánea

_CONTEXTS Ryu

_CONTEXTS es un diccionario para especificar los contextos que se van a utilizar en la aplicación RYU. La clave es el nombre del contexto y su valor es un clase normal que implementa el contexto.

La clase es instanciada por app_manager y la instancia es compartida entre las subclases RyuApp que tengan miembros _CONTEXTS con la misma clave. Una subclase RyuApp puede obtener una referencia a la instancia utilizando su __init__ kwargs como se muestra a continuación.

En esta implementación se ha utilizado un diccionario que relaciona direcciones ip con direcciones MAC, para poder compartir este diccionario en los distintos programas se han utilizado los mencionados contextos.

```
_CONTEXTS = {'ip_mac' : contextos.ip_mac}


self.ip_mac = kwargs['ip_mac'].ip_mac
```

Capítulo 4

Resultados

Se ha conseguido generar topologías de cualquier tamaño y disposición pudiendo controlar que privilegios tiene cada usuario.

El primer caso que se muestra es el caso más básico, un switch sencillo con sólo dos hosts.

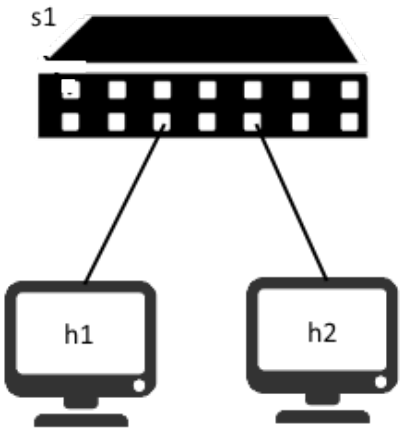


Figura 4.1: Topología en estrella básica

Como se ve en esta primera figura no hay conexión entre los dos terminales.

```
mininet> h1 ping h2
PING 10.0.0.11 (10.0.0.11) 56(84) bytes of data.
^C
--- 10.0.0.11 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 2999ms
```

Figura 4.2: Prueba de conexión

Como se ve en la siguiente imagen el usuario 1 pertenece al grupo 1 y el usuario 2 pertenece al grupo 2.

user	password	uid	gid
Filter	Filter	Filter	Filter
1	1	1	1
2	2	2	2

Figura 4.3: Tabla Users.

Por lo tanto a ambos usuarios se les aplicará la política 1 como se muestra en esta figura.

gid1	gid2	policy
Filter	Filter	Filter
2	1	1
1	2	1

Figura 4.4: Tabla Connections.

La política 1 acepta cualquier tráfico como se indica a continuación.

policy	protocol	o_port	d_port	action	priority
Filter	Filter	Filter	Filter	Filter	Filter
1				ACCEPT	10

Figura 4.5: Tabla Policies.

Al logear los dos hosts ya sea accediendo al portal web o mediante el comando:

```
h1 curl --insecure -X POST -d 'login=1&password=1'
     https://10.0.0.1:8080/login
```

donde h1 indica el host desde donde se ejecuta el comando, el login y el password son las credenciales del usuario y la dirección 10.0.0.1 le corresponde al autenticador, se recogen las políticas correspondientes a dicho usuario para aplicarlas en los switches. Cabe destacar que al intentar establecer conexión entre los hosts es cuando realmente se aplican las reglas de flujo.

```
mininet> h1 curl --insecure -X POST -d 'login=1&password=1' https://10.0.0.1:808
0/login
<h1>You are logged in</h1><a href="/logout">Logout</a>mininet>
mininet>
mininet>
mininet> h2 curl --insecure -X POST -d 'login=2&password=2' https://10.0.0.1:808
0/login
<h1>You are logged in</h1><a href="/logout">Logout</a>mininet>
mininet>
mininet>
mininet> h1 ping h2
PING 10.0.0.11 (10.0.0.11) 56(84) bytes of data.
64 bytes from 10.0.0.11: icmp_seq=1 ttl=64 time=61.3 ms
64 bytes from 10.0.0.11: icmp_seq=2 ttl=64 time=0.452 ms
64 bytes from 10.0.0.11: icmp_seq=3 ttl=64 time=0.125 ms
^C
--- 10.0.0.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.125/20.652/61.379/28.798 ms
```

Figura 4.6: Conexión entre h1 y h2

Se puede ver como el controlador devuelve AUTENTICADO cuando el login se ha realizado correctamente.

```
loading app arp.py
loading app dhcp.py
loading app permisos.py
loading app contextos.py
loading app switch.py
loading app ryu.controller.ofp_handler
creating context mac_to_port
creating context ip_mac
instantiating app arp.py of ARPHandler
instantiating app permisos.py of AUTHCo
instantiating app switch.py of SimpleSw
instantiating app ryu.controller.ofp_ha
instantiating app dhcp.py of DHCPRespon
AUTENTICADO
AUTENTICADO
if de policies
INSERTA POLITICA: 10.5.0.10 10.6.0.11
if de policies
INSERTA POLITICA: 10.6.0.11 10.5.0.10
```

Figura 4.7: Respuesta del controlador

La siguiente figura muestra las reglas de flujo insertadas en el switch s1, las reglas marcadas son las que intervienen en la conexión entre h1 y h2 que siguen los pasos descritos en 3.2.3

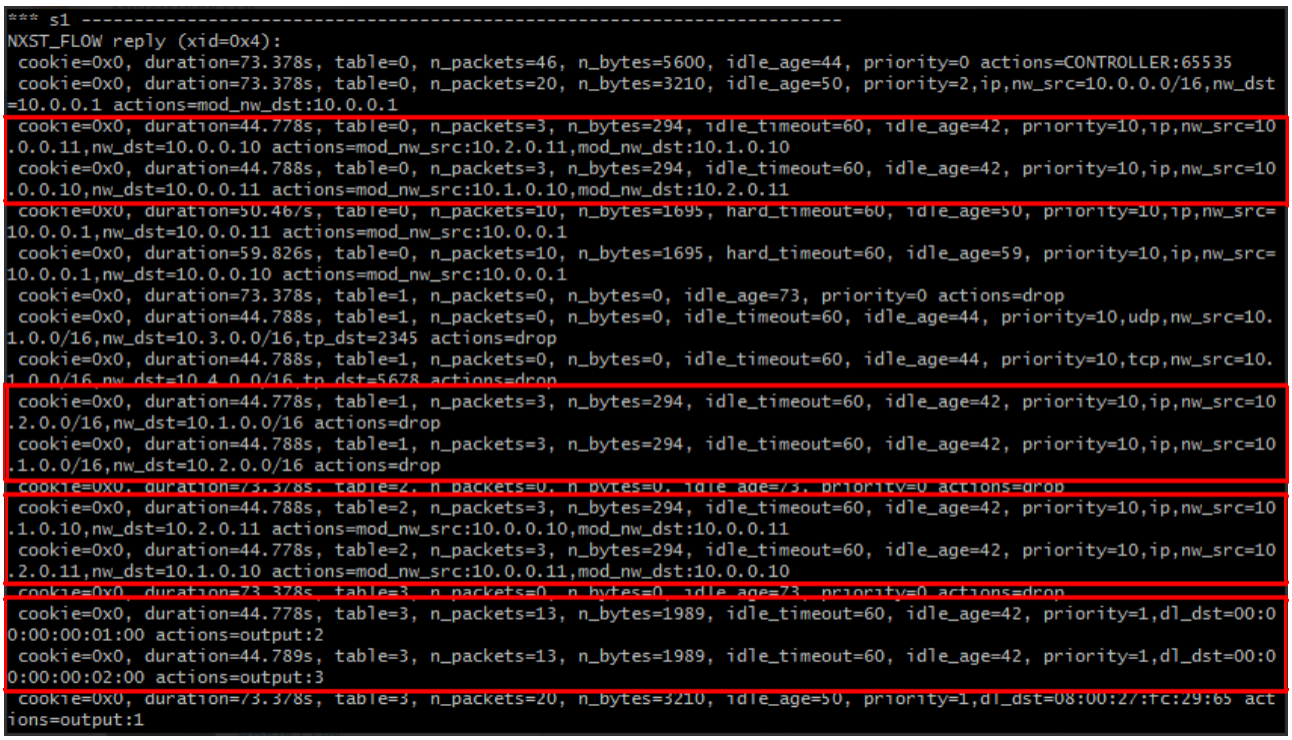


Figura 4.8: Reglas de flujo aplicadas al switch s1

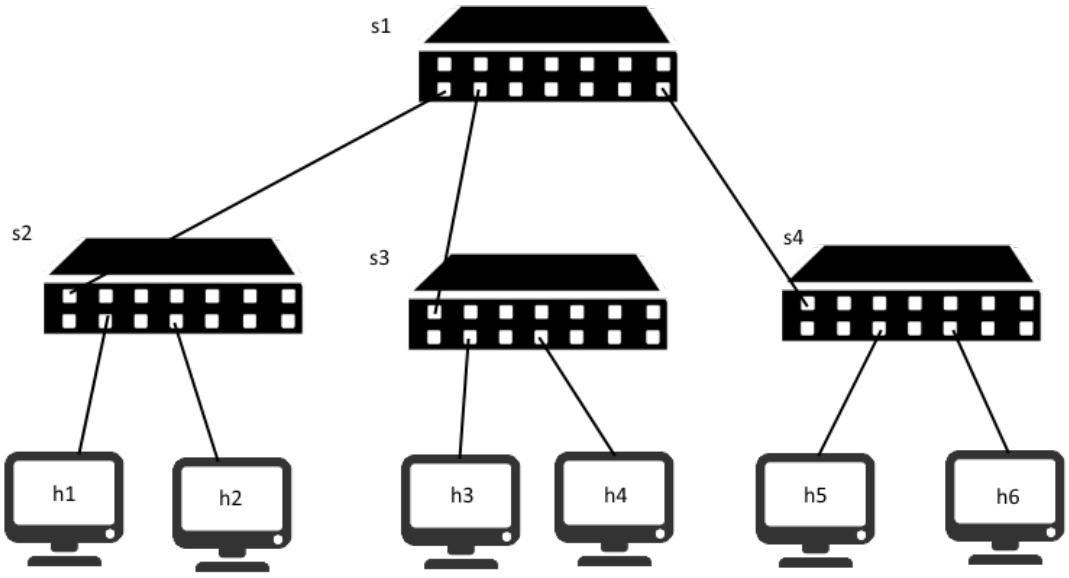


Figura 4.9: Topología en árbol

Esta segunda topología es algo más compleja contiene 6 hosts, 4 switches y tiene forma de árbol, en este caso se van a realizar envíos TCP.

Análogamente al caso mostrado anteriormente, la política 3 será aplicada a la conexión entre usuarios del grupo 1 con usuarios del grupo 4 y la política 5 en el caso inverso, del grupo 4 al grupo 1.

policy	protocol	o_port	d_port	action	priority
Filter	Filter	Filter	Filter	Filter	Filter
1				ACCEPT	10
2	udp		2345	ACCEPT	10
3	tcp		5678	ACCEPT	10
4	udp	2345		ACCEPT	10
5	tcp	5678		ACCEPT	10

Figura 4.10: Tabla Policies conexión TCP.

Esta tabla define tanto la política 3 como la 5, la 3 indica que permite conexiones TCP con puerto de destino 5678 y la 5 indica que permite conexiones TCP con puerto de origen 5678.

Por lo tanto el host 2 (h2), se pondrá en modo escucha con el puerto 5678 ya que este usuario se le está aplicando la política 5; y el h1 será quien mande el paquete a h2 con puerto de destino 5678.

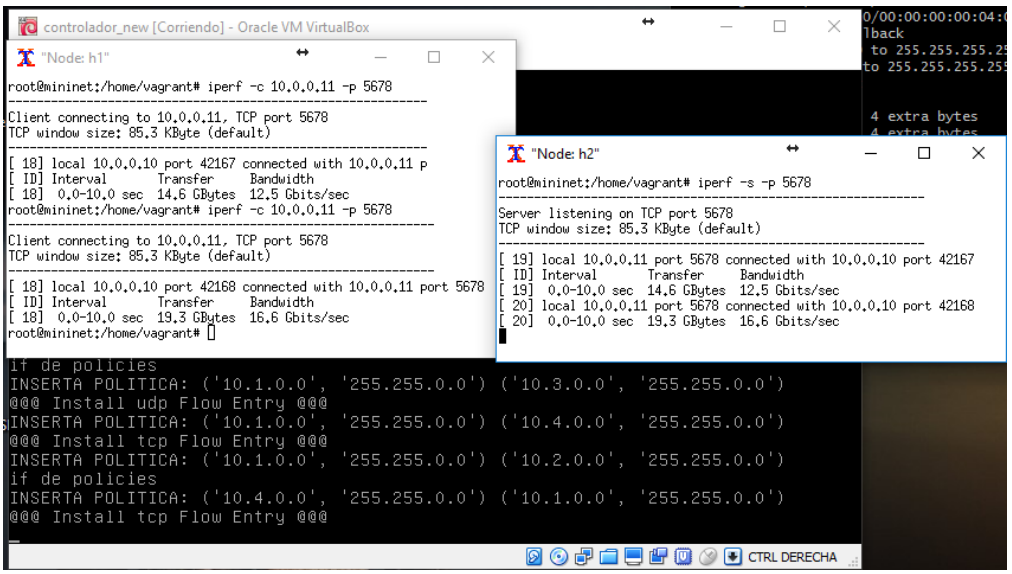


Figura 4.11: Envío paquete TCP

Aparte de políticas a nivel de grupo también se pueden establecer a nivel de usuario, que por lo general tendrán mas prioridad que las primeras, para denegar o conceder determinados privilegios a ciertos usuarios.

Este último caso es para mostrar que las políticas se pueden aplicar a nivel de usuario en h3 se va a logear el usuario 5 y en h5 se va a logear el usuario 6 quienes no tienen grupo asignado, pero tienen la política asignada para la conexión entre sí.

uid1	uid2	policy
Filter	Filter	Filter
5	6	1
6	5	1

Figura 4.12: Usuarios sin grupo.

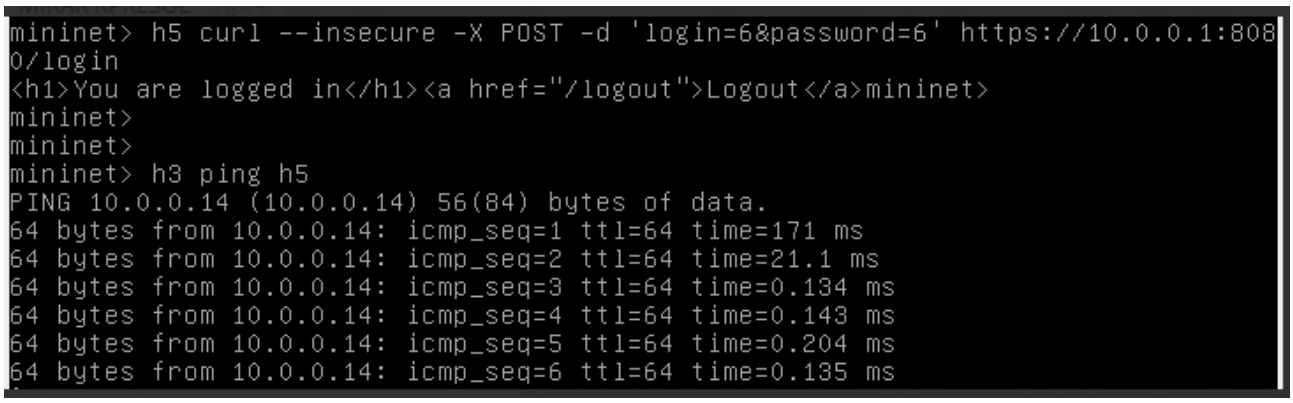


Figura 4.13: Conexión entre h3 y h5.

Gracias a esta implementación usuarios dentro de una misma red se puedan comunicar entre sí en base a que grupo pertenece y con el uso de switches OpenFlow, usuarios de redes distintas se pueden conectar entre sí mediante switches, sin tener que hacer uso de un router para interconectar distintas redes.

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

Tras la realización de este trabajo de fin de grado se han obtenido las conclusiones que se enumeran a continuación:

- Se ha implementado el software de control para una arquitectura de red basada en SDN. Dicho software controla una red plana, sin subredes, distinguiendo distintos grupos de usuarios que podrán conectarse entre sí o no según una serie de políticas, permitiendo así un control de la conectividad basado en usuarios.

- La implementación de este software ha permitido aprender los conceptos relacionados con esta tecnología, así como la adquisición de las destrezas prácticas necesarias para el manejo de un software controlador, en este caso Ryu.

- Se han diseñado e implementado, utilizando el simulador Mininet, los escenarios de prueba necesarios para la depuración del software de control.

- Se ha implementado mediante Python un módulo de autenticación sencillo, que ha permitido la realización de las pruebas, aunque en una implementación real éste deberá evolucionarse o sustituirse por otro.

- La realización de este trabajo ha permitido conocer el funcionamiento básico de OpenFlow, que es uno de los protocolos estándar dentro de SDN.

- Al tratarse de una red plana a nivel de enlace, se ha tenido que modificar el comportamiento de la red comparado con las redes convencionales. El tráfico broadcast no se puede reenviar de forma indiscriminada. Por ello,

este tipo de tráfico generado por los protocolos ARP y DHCP, ya no se maneja como tráfico broadcast, sino que las respuestas para estos protocolos son gestionadas de forma centralizada (controlador) y los paquetes broadcast ya no llegan a todos los hosts conectados. Habría que proceder del mismo modo para otros protocolos basados en broadcast.

- De igual modo, el comportamiento de los switches de capa 2 también ha sido adaptado. Se suprime el autoaprendizaje y la tabla de reenvío de los switches se obtiene del listado hosts que han solicitado dirección IP mediante DHCP. El inconveniente es que dos hosts sin dirección IP nunca podrán comunicarse a nivel de enlace. Esto habrá que tenerlo en cuenta cuando se usen protocolos que no estén basados en IP. Desde el punto de vista de la seguridad, sólo se podrán enviar tramas entre hosts conocidos, lo que es una ventaja.
- El esquema de conectividad basado en usuarios permite la reconfiguración dinámica de la red según los usuarios conectados.

5.2. Trabajos futuros

Las líneas de trabajo futuras que pueden servir para mejorar este proyecto son:

- Mejorar o sustituir el sistema de autenticación.
- Probar el sistema en un entorno más realista y realizar una auditoría de seguridad.
- Revisar si es necesario dar soporte a algún protocolo basado en broadcast, que no se haya contemplado.
- Hacer pruebas en una red con hardware real, realizando tests de carga.

Capítulo 6

Summary and Conclusions

6.1. Conclusions

- Control software for an SDN-based network architecture has been implemented. This software controls a flat network, without subnets, distinguishing different groups of users that can be connected to each other or not according to a series of policies, allowing a control of the user-based connectivity.
- The implementation of this software has allowed to learn the concepts related to this technology, as well as the acquisition of the practical skills necessary to handle a software controller, in this case Ryu.
- The test scenarios necessary for the debugging of the control software have been designed and implemented using the Mininet simulator.
- A simple authentication module has been implemented through Python, which has allowed the tests to be performed, although in an actual implementation it must be evolved or replaced by another one.
- The realization of this work has allowed to know the basic operation of OpenFlow, which is one of the standard protocols within SDN.
- As it is a flat network at the link level, it has had to modify the behavior of the network compared to conventional networks. Broadcast traffic can not be forwarded indiscriminately. Therefore, this type of traffic generated by the ARP and DHCP protocols is no longer handled as broadcast traffic, but the responses for these protocols are managed centrally (controller) and the broadcast packets no longer reach all connected hosts. It should be done in the same way for other broadcast-based protocols.
- Similarly, the behavior of Layer 2 switches has also been adapted. Self-learning is suppressed and the switch's forwarding table retrieves from the list hosts that have requested IP address through DHCP. The drawback is

that two hosts without an IP address will never be able to communicate at the link level. This will have to be taken into account when using protocols that are not based on IP. From the point of view of security, only frames between known hosts can be sent, which is an advantage.

- The user-based connectivity scheme allows the dynamic reconfiguration of the network according to the connected users.

6.2. Future work

The future lines of work that can serve to improve this project are:

- Improve or replace the authentication system.
- Test the system in a more realistic environment and perform a security audit.
- Check if it is necessary to support some protocol based on broadcast, that has not been contemplated.
- Perform load tests in a network with real hardware .

Capítulo 7

Presupuesto

En la siguiente tabla, se muestra el presupuesto de ejecución de este trabajo de fin de grado.

Objeto	Coste
Licencias de Software	0 €
Amortización de equipos	40 €
Salario (150 horas x 26€)	3.900 €
Subtotal	3.940 €
IGIC 7%	275,80 €
Total	4.216 €

Figura 7.1: Presupuesto

Bibliografía

[1] Desconocido. Historia y evolución de las redes. https://fumsnuevastecnologias.wordpress.com/2011/10/05/historia-de-las-redes/, 2011.

[2] Dr. Chih-Heng Ke. ShortestPathFirstSwitch. http://csie.nqu.edu.tw/smallko/sdn/ryu_sp13.htm, 2011.

[3] Himanshu Arora. What is DHCP and How DHCP Works? http://www.thegeekstuff.com/2013/03/dhcp-basics/, 2013.

[4] Inycom. SDN (Software Defined Networking): la gran revolución de la red. http://www.inycom.es/actualidad-informatica/opinion-tic/480-sdn-software-defined-networking-la-gran-revolucion-de-la-red, 2015.

[5] Jim Frey. SDN Use Cases: Myths and Realities. http://blog.radware.com/sdn/2015/02/sdn-myths-and-realities/, 2015.

[6] J.P. Lüke. Práctica nº 11, Laboratorio de Redes. Guión de prácticas. Laboratorio de Redes. Grado de Ingeniería Informática. Curso 2014-15. Universidad de La Laguna.

[7] Mininet Team. Mininet Overview. http://mininet.org/overview/, 2016.

[8] Nick Feamster and Jennifer Rexford and Ellen Zegura. The Road to SDN - An intellectual history of programmable networks. http://queue.acm.org/detail.cfm?id=2560327, 2013.

[9] Open Networking Foundation. Openflow switch specification. http://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[10] Open Networking Fundation. Software-Defined Networking: The New Norm for Networks. https://www.opennetworking.org/images/

stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, 2012.

[11] Oracle, Inc. Virtualbox. https://www.virtualbox.org/, 2016.

[12] Python. What is Python? https://www.python.org/doc/essays/blurb/, -.

[13] Ryu. SimpleSwitch. https://github.com/osrg/ryu/blob/master/ryu/app/simple_switch_13.py, 2015.

[14] SDN Hub. RYU Controller Tutorial. http://sdnhub.org/tutorials/ryu/, 2016.

[15] SQLite project. About SQLite. https://sqlite.org/about.html, 2016.

[16] The CherryPy team. CherryPy. http://tools.cherrypy.org/wiki/sp/Tutorial, 2016.