eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ Trabajo Fin de Grado ▪

Ingeniería de Computadores

Control de Acceso vía SDN para redes IoT

Jon Ander Sukia

2021

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ Trabajo Fin de Grado ▪

Ingeniería de Computadores

Control de Acceso vía SDN para redes IoT

Jon Ander Sukia

2021

Dirección

José Miguel-Alonso

Resumen

Que los dispositivos IoT (Internet of Things) tengan acceso ilimitado a Internet proporciona muchas ventajas, pero también puede convertirse en un riesgo dada la naturaleza maliciosa de ciertos programas o lugares de la red. Es por eso, entre otras razones, que nos vemos en la obligación de controlar o limitar el acceso de estos dispositivos en nuestras redes, para poder aislar cualquier tipo de potencial problema o amenaza y que no perjudique el resto de la red.

El objetivo de este trabajo es crear un sistema NAC (Network Access Control) sencillo, adecuado a entornos IoT. Para hacerlo, nos basamos en la capacidad que tienen las SDN (Software Defined Networks) para programar el comportamiento de los dispositivos de red. Queremos crear una herramienta intuitiva de cara al usuario prescindiendo de la complejidad de una infraestructura de clave pública y de gestión de credenciales.

Unlimited access to the Internet for IoT (Internet of Things) devices can provide a plethora of benefits, but it can also become a risk due to the malicious nature of some websites and malware. This is the main reason why we must control and limit the access of the equipment in our system, in order to isolate any possible threat or issue that might damage or compromise the rest of it.

The aim of the project is to create a simple NAC (Network Access Control) for IoT environments. In order to achieve it, we take advantage of the ability to program the behavior of network devices provided by SDN (Software Defined Networks). We want to create an intuitive tool for the end user avoiding the complexity that credential management and a public key infrastructure brings to the table.

IoT (Internet of Things) gailuek Interneterako sarbide mugagabea izateak abantaila asko ekar ditzake, baina, era berean, arrisku bat izan liteke sareko zenbait programa edo tokiren izaera maltzurra dela eta. Horregatik, beste arrazoi batzuen artean, gailu horiek gure sareetan duten sarbidea kontrolatu edo mugatu behar dugu, edozein arazo edo mehatxu mota isolatu ahal izateko eta sarearen gainerakoa babesteko.

Lan honen helburua IoT inguruneetara egokitutako NAC (Network Access Control) sinple bat sortzea da. Horretarako, SDNek (Software Defined Networks) sareko gailuen portaera programatzeko duten gaitasunean oinarritzen gara. Tresna intuitibo bat sortu nahi dugu erabiltzaileari begira, gako publikoko eta kredentzialak kudeatzeko azpiegitura baten konplexutasuna alde batera utzita.

Índice

Capítulo 1. Introducción y objetivos

Con el paso del tiempo, el uso de dispositivos IoT (Internet of Thigs) como móviles u ordenadores se ha vuelto parte de nuestra vida cotidiana y rutina diaria. Estos dispositivos se encuentran en una situación constante de intercambio de datos gracias al acceso a Internet, lo cual es muy positivo ya que tenemos cantidades ingentes de información al alcance de nuestra mano, pero la otra cara de la misma moneda nos hace ver que también existen programas o lugares con intencionalidad maliciosa que pueden comprometer nuestros dispositivos. Es por eso por lo que mencionamos la necesidad de controlar o limitar el acceso de dichos dispositivos a la red o a internet.

El objetivo de este TFG es crear un software capaz de gestionar los dispositivos de una red de forma que podamos controlar cómo y con qué dispositivo o grupo de dispositivos se puede comunicar cada uno de ellos. La particularidad es que el sistema NAC (Network Access Control) que vamos a crear es que será llevado a cabo en el contexto de una red definida por software (SDN) basada en el estándar OpenFlow.

En la *Ilustración 1* podemos observar el esquema general del TFG. La finalidad de este proyecto es trabajar con switches y hosts reales, ya sean físicos o virtuales (por ejemplo, en un entorno Cloud). Como no disponemos de dicho entorno, utilizaremos Mininet para poder emular de forma 100% realista este tipo de redes con distintas características, compatibles lógicamente con la especificación OpenFlow. Por otro lado, deberemos crear un formato ACL (Access Control List) con que sea intuitivo para el administrador de la red (el "usuario"), de forma que sea posible crear un fichero con las reglas que queramos añadir (ej. PC1 puede comunicarse con PC2, y sólo puede comunicarse con PC3 vía ICMP). Por lo tanto, nuestra aplicación NAC deberá ser capaz de leer dicho fichero, e interpretar las reglas escritas de tal forma que se comunique con los controladores de la red implementando las reglas de forma adecuada.

Además, tener el proceso inverso también resulta útil, contando así con la opción de extraer las reglas que están siendo aplicadas en un momento dado y crear un fichero de configuración a partir de ellas (con el formato de la ACL mencionado previamente). De esta forma podemos verificar que la configuración aplicada es realmente la deseada, guardar los cambios realizados, y usar el nuevo fichero para poder configurar de nuevo con la red (gracias a la funcionalidad principal comentada previamente) o simplemente almacenarlo en forma de back-up.

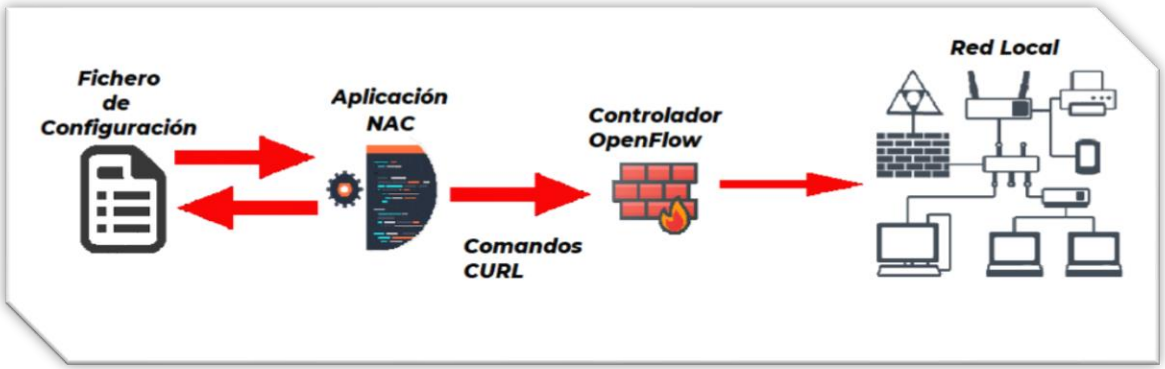

Ilustración 1 Esquema de la estructura del TFG [I_1]

1.1 Estructura de la memoria

Una vez aclarado el objetivo del trabajo de fin de grado, comenzaremos a explicar los términos previamente mencionados que componen el proyecto, para entender qué son y cómo van a ser utilizados, como por ejemplo un NAC. Este será el objetivo del segundo capítulo, y le seguirá el tercero el cuál abordará el entorno y las herramientas que utilizaremos para poder desarrollar el TFG, además de aportar información a cerca de los conceptos más destacados de estas herramientas. Asimismo, el cuarto capítulo será destinado a compartir la implementación del programa desarrollado y las funcionalidades añadidas, compartiendo a su vez decisiones tomadas a lo largo de la creación de este.

Por otro lado, también será destinado una sección para compartir las conclusiones sacadas a lo largo y al final del trabajo, y en el cual se señalarán también las funcionalidades que sería posible añadir en un futuro. Este apartado tiene especial interés ya que contamos con ideas interesantes en caso de querer seguir desarrollando la aplicación. El siguiente capítulo se dedicará a exponer la planificación seguida para completar este TFG, y las desviaciones que hayan podido surgir. Por último, contaremos con la bibliografía utilizada en este proyecto y con un anexo, en el cual se facilitarán pautas para poder replicar y poner en marcha este TFG, desde el software necesario, comandos que utilizamos de manera frecuente y el propio código de la aplicación.

Capítulo 2. Contexto: NAC, ACL, SDN y OpenFlow

2.1 NAC

2.1.1 Definición

Las siglas NAC provienen de "Network Access Control", que significa control de acceso a red. El objetivo del NAC es exactamente lo que implica su nombre, ser una solución que utiliza diferentes mecanismos y protocolos para definir e implementar una política que describa qué actividades de comunicación están permitidas para un nodo conectado a la red. De esta forma nos aseguramos de que los hosts que accedan a la red cumplen con los requisitos preestablecidos y, en caso de no cumplir con ellos, se puede integrar un proceso de remedio automático el cual corrige los nodos que no cumplen las reglas antes de permitir su acceso.

Un buen ejemplo sería cuando un PC se conecta a la red. En este momento el dispositivo no tiene acceso a nada a no ser que cumpla con las políticas establecidas. Una vez se comprueba que el nodo cumple dichas políticas o requisitos, se le concede acceso de acuerdo con lo definido con el sistema NAC. Por ejemplo, se le podría permitir comunicarse únicamente con PC-s de su mismo departamento y con un servidor que almacena datos relacionados con su departamento.

2.1.2 Objetivos

Los sistemas NAC representan una categoría emergente dentro de la seguridad de redes, y por eso tanto su definición como sus propósitos evolucionan con el paso del tiempo. A continuación, enumeramos los principales objetivos de un sistema NAC:

- Mitigar los "Zero-day-attacks": Estos ataques comprometen el sistema e incluso podrían obtener información sensible de los trabajadores o usuarios de un producto o servicio.

- Reforzar las políticas de seguridad corporativas: Como hemos mencionado previamente, un NAC permite crear políticas y forzarlas en los dispositivos de red (routers, switches) para permitir o denegar el acceso de ciertos nodos dependiendo de la circunstancia y de las necesidades de cada red.

- Administración de acceso e identidad: Los dispositivos NAC refuerzan las políticas de acceso basándose en identidades de usuarios autenticados (para

usuarios finales como PCs) en vez de basarse en direcciones MAC o IP, que pueden ser fácilmente falsificadas.

2.1.3 Conceptos

- Pre-admisión vs. post-admisión: Por un lado, se considera pre-admisión el hecho de inspeccionar los nuevos hosts antes de entrar a la red o cuando son incorporados a la misma. Mientras son inspeccionados no podrán acceder a ella ni a su contenido. A modo de ejemplo, se puede prevenir que un equipo con el antivirus desactualizado se pueda conectar a servidores sensibles: este es un caso de pre-admisión. Por otro lado, la post-admisión se basa en otorgar decisiones de refuerzo una vez el dispositivo en cuestión haya accedido a la red.

- Con agente vs. sin agente: La idea fundamental sobre NAC es habilitar la red a tomar decisiones sobre el control de acceso basándonos en información obtenida del "end-system" o host. Por esta razón, la manera en la que se informa a la red sobre un dispositivo es muy relevante. La diferencia clave es si necesitan un software o agente externo que informe a la red sobre el nuevo dispositivo y sus características, o si escanean la red o utilizan algún tipo de técnica para realizar un inventario de la red para detectar y notificar un nuevo host.

- Fuera de banda vs. en línea: En sistemas fuera de banda se distribuyen agentes a los hosts, los cuales informan a una consola central la cual a su vez controla los switches y puede implementar políticas adecuadas. Por otro lado, la solución en línea es contar con un sistema interno de firewall para redes de capa de acceso para forzar las políticas establecidas. Las soluciones de fuera de banda tienen la ventaja de reutilizar infraestructura existente, mientras que la solución en línea es más sencilla de desplegar en redes nuevas. Sin embargo, hay algunos productos que cuentan con ambas ventajas sobre una instalación más sencilla y menos peligrosa, siendo fuera de banda, pero con técnicas para aportar la eficacia de en línea ante dispositivos desobedientes.

- Cuarentena y portal cautivo: Cuando se despliega un sistema NAC se da por hecho que habrá usuarios o hosts legítimos que se vean con el acceso denegado, ya que es frecuente tener el antivirus desactualizado entre otras razones. Por eso el NAC debe de contar con algún mecanismo para remediar estos problemas, que causan que se les deniegue el acceso a los usuarios. Estas son las dos estrategias más comunes:

    - Cuarentena: Una red de cuarentena es una red IP restringida por acceso a ciertos hosts y aplicaciones. Suele ser implementada usando una distribución de VLAN. Cuando el NAC detecta que un usuario está desactualizado, el puerto es asignado a cierta VLAN que le pondrá en contacto con un servidor de actualización y parcheado.

    - Portales cautivos: Un portal cautivo intercepta acceso a webs vía HTTP que redirige al usuario a una aplicación web que divulga instrucciones

sobre cómo actualizar su dispositivo. Hasta que el host pase inspecciones automáticas no se le dará acceso a nada más aparte del portal cautivo.

2.1.4 Ejemplo y reflexión

OpenNAC es un software NAC "open source" orientado a redes corporativas LAN/WLAN. Habilita autenticación, autorización y la auditoría basada en políticas de acceso a la red. Una de las ventajas de este producto es que es fácil de incorporar a sistemas existentes y es flexible con extensiones y nuevas características. Además, cuenta con un sistema de detección de actualizaciones de sistemas operativos, antivirus y firewalls de dispositivos conectados para hacer cumplir las reglas definidas para permitir el acceso a la red.

El objetivo de este proyecto es distinto al de soluciones NAC complejas como OpenNAC, ya que vamos a trabajar en un contexto IoT en el cual no exigiremos potencia de cálculo a los equipos conectados y trabajaremos a nivel de switch para gestionar nuestra propia NAC, como veremos más adelante. Se tratará de un sistema NAC de pre-admisión ya que inicialmente ningún dispositivo puede comunicarse, y sólo podrán hacerlo si alguna regla de la ACL así lo permite. Por definición será una red con agente ya que los switches identificarán la presencia de nuevos nodos e informarán de ello al controlador, que gestionará el comportamiento a seguir basándose en la ACL. Además, se considerará una NAC en línea ya que, a pesar de no tener un firewall como tal, gracias al controlador OpenFlow y la implementación de una API de RYU añadiremos la funcionalidad de un firewall orientado a forzar las reglas establecidas como veremos más adelante. Por último, en nuestro caso no planteamos una situación de cuarentena o portales cautivos ya que no diferenciaremos el acceso dependiendo del estado del software de los nodos conectados.

En este proyecto nos centramos íntegramente en los niveles de red y de switch, es decir, que las decisiones sobre si un paquete avanza o no se deciden en base a la información de destinatario y origen (ya sea dirección IPV4/V6, dirección MAC, puerto físico) y otras características como el protocolo, las cuales serán expuestas en el apartado de implementación.

2.2 ACLs

2.2.1 Definición

Una "Access Control List" (ACL), es una lista con permisos asociados a recursos del sistema. Permite controlar el flujo del tráfico en equipos de una red (p. ej. en switches). Su principal objetivo es filtrar el tráfico de la red gestionando qué paquetes son admitidos en la red y cuales son bloqueados.

Para determinar qué paquete avanza y cual es denegado necesitamos establecer una base de reglas. Una parte de cada regla recoge la forma de identificar el paquete o el dispositivo, por ejemplo, una dirección IP o MAC, y la acción a realizar una vez haya un "match" con alguno (denegarlo o dejar que pase).

2.2.2 Propósito

El rápido crecimiento de la tecnología de redes acarrea desafíos para la seguridad de redes y el QoS (Quality of Service). Utilizar un firewall o un NAC apoyándonos en un formato de ACL nos ayuda a reforzar la red para prevenir los siguientes problemas:

- Prevenir filtraciones de información y accesos no autorizados a recursos en servidores clave de la red.

- Para prevenir virus entrando y comprometiendo la red.

- Para prevenir que algunos servicios acaparen más ancho de banda de lo debido, de esta forma asegurando el ancho de banda para procesos más sensibles a las demoras o a las pérdidas de paquetes como pueden ser audios o vídeos.

En el contexto de este TFG, el propósito es darle una herramienta al usuario con un formato intuitivo para poder especificar el tipo de comunicación permitido entre nodos de la red. Para ello utilizará las reglas convenientes en el formato establecido y nuestro software procesará esas reglas e implementará los flujos correspondientes en el controlador.

2.2.3 Formato

A continuación, vamos a revisar un formato básico de ACL. Nos basaremos en el ejemplo de la Ilustración 2. Para empezar, tenemos el Rule y el Rule ID los cuales nos muestran el número asociado a dicha regla, y al igual que señala el nombre, le sigue la acción que se realizará si hay un match en las condiciones, permitir el paso de dicho paquete o bloquearlo. A continuación, tenemos la dirección IP, que podría ser también una dirección MAC, y dependiendo del caso tendremos la procedencia y el destinatario. Además, contamos también con una regla "catch all", es decir, una regla final a la que llegamos de forma explícita o por omisión del resto, la cual se encarga dar una respuesta para todos aquellos paquetes que no se han emparejado con las reglas establecidas hasta el momento. En este caso (en la imagen: "rule 4294967294 deny") cualquier paquete que no se haya emparejado con las reglas anteriores sí lo hará con esta y será bloqueado. Esto dependerá de la política de acceso que se quiera seguir, en este caso se denegará todo exceptuando los paquetes que coincidan con las reglas, pero hay otras redes que siguen la política opuesta.

Basándonos en este tipo de formato diseñaremos uno que satisfaga las necesidades de nuestra red y poder gestionar la red con el controlador. Partiendo de este ejemplo se puede expandir y añadir muchos otros requisitos o especificaciones, como por ejemplo añadir también puertos o protocolos específicos. En el apartado de implementación podremos observar el formato concreto de ACL desarrollado para el proyecto.

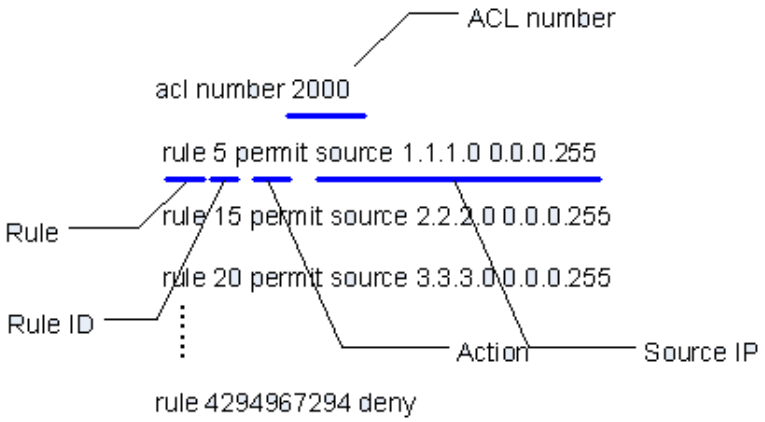

Ilustración 2 Estructura del ACL [I_2]

De manera análoga, la Ilustración 3 nos muestra cómo se realiza la lectura y la comprobación de la ACL cuando llega un paquete. Los dos primeros requisitos se aseguran de que la ACL exista y de que haya reglas dentro del mismo. Asimismo, se comprobará en orden si hay alguna regla que empareje con el paquete recibido. Si la hay, se comprueba si la regla permite que el paquete continúe su trayecto o si ha de ser bloqueado. Del mismo modo, si llega un momento en el que no hay más reglas se indicará que el paquete no cumple ninguna de las reglas (al igual que si no hay reglas o si no existe la ACL).

Importante acentuar que en el momento que se encuentra una regla que haga "match" no se sigue buscando otra, por lo tanto, debemos tener esto en cuenta a la hora de escribir cualquier fichero con reglas de cara a una ACL, ya que sólo se aplicará el primer match. Teniendo esto en cuenta debemos escribir reglas de más concretas al inicio y más genéricas al final. Por ejemplo, en el caso de querer habilitar toda la comunicación del PC1 al PC2 exceptuando la comunicación ICMP, primero debería de aparecer una regla denegando paquetes ICMP de PC1 a PC2, y otra regla después de esta que habilite toda comunicación entre ambos. Si se hace al revés primero se encontrará la regla que habilita la comunicación del PC1 al PC2 y no buscará el siguiente match que sería el de bloquear ICMP, así que también dejaría pasar las tramas ICMP.

Por otro lado, como ya hemos comentado previamente, debemos de tener una regla final la cual se encargue de dar una respuesta a los paquetes que no hagan "match" con ninguna de las reglas. En este caso, la Ilustración 3 mencionaba que si no hay más reglas y el paquete no ha hecho match hasta entonces, se bloquea el paquete. Pues bien, esto se recoge en una regla final, la cual podría ser la opuesta dependiendo de la política de acceso, es decir, que podríamos tener reglas para denegar ciertos paquetes y en el caso de que no sea uno de ellos aceptarlo, pero tanto en el ejemplo de la ilustración como en nuestra ACL, se seguirá esta política de denegar todos los paquetes que no sean emparejados. Esta regla viene ya incluida por defecto al iniciar el Rest_Firewall, por lo tanto no es necesario especificarla en nuestra ACL, pero es un concepto importante a tener en cuenta que sí que puede ser necesario en otros formatos o implementaciones ajenos a este TFG.

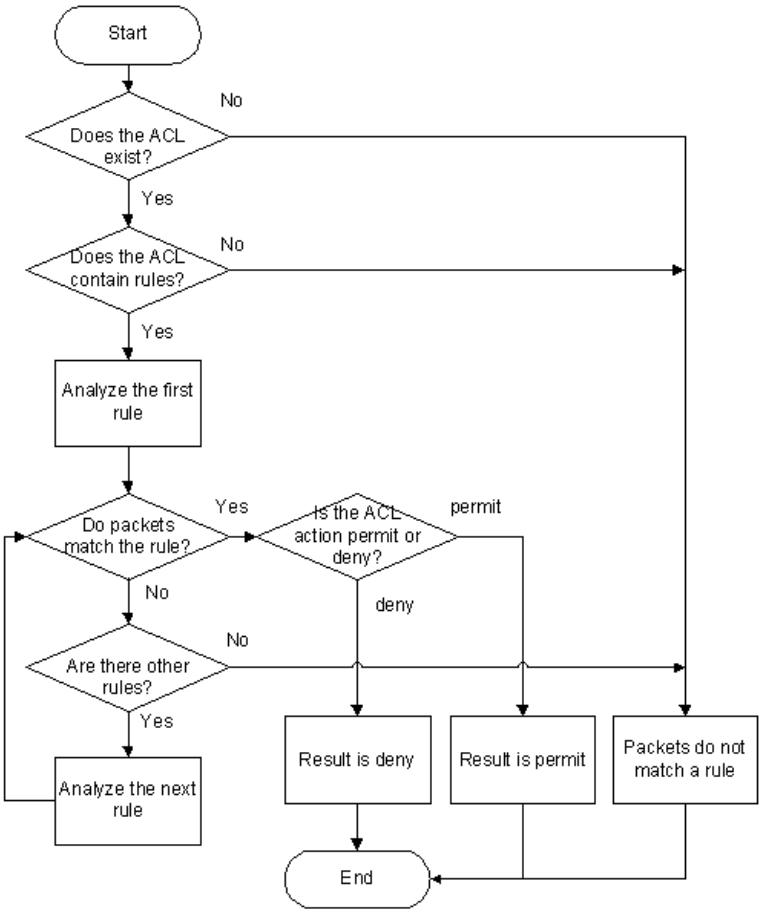

Ilustración 3 Mecanismo de Match de un ACL [I_3]

2.3 SDN

2.3.1 Definición

El Software-Defined Networking (SDN) habilita una configuración de la red dinámica y eficiente para mejorar el rendimiento de esta. Las arquitecturas SDN desacoplan el control de la red, de las funciones de reenvío, habilitando que el control de acceso sea programable y que la estructura subyacente sea abstraída de aplicaciones y servicios de red. También puede aportar la monitorización y la flexibilidad que necesitan las infraestructuras de hoy en día. Los SDN centralizan la "inteligencia" de la red en uno o más controladores que son los que gestionan con la información que tienen lo que se hará con los paquetes de datos intercambiados entre los hosts. SDN suele ser asociada con el protocolo OpenFlow, con el que vamos a trabajar en este proyecto. Sin embargo, hay otras formas de trabajar con el mismo concepto sin usar OpenFlow.

2.3.2 Componentes de la arquitectura

- Aplicación SDN: Las aplicaciones SDN son programas que comunican directamente sus requisitos y el comportamiento deseado de la red al controlador SDN vía interfaces norte (NBI). Además, pueden gestionar una visión abstracta de la red y ejecutar sus propias decisiones internas.

- Controlador SDN: El controlador SDN es una entidad centralizada que se encarga de (i) traducir los requisitos de la aplicación SDN a los Datapath SDN, (ii) proveer una vista abstracta de la red a la aplicación SDN (que pueden incluir estadísticas y eventos). Un controlador SDN consiste en uno o más agentes SDN, el control lógico SDN, y la interfaz SDN de control a Data-Plane (CDPI).

- Datapath (Ruta de datos) SDN: El datapath SDN es un componente lógico de red el cual expone visibilidad y controla la capacidad de procesamiento y reenvío de datos. Un datapath SDN está formado por un agente CDPI, uno o más motores de reenvío, y cero o más funciones de procesamiento de tráfico. Esto último puede incluir reenvíos simples entre interfaces externas y el procesamiento interno del tráfico. En el contexto de este TFG, el rol de datapath lo ejecutarán switches compatibles con OpenFlow. Para este proyecto usaremos switches virtuales Open vSwitch [13]. Los hosts (ordenadores, dispositivos IoT) se conectan a la red mediante algunos de estos switches.

- Interfaz SDN de control a Data-Plane (CDPI): SDN CDPI es la interfaz definida entre un controlador SDN y un datapath (switch) SDN, el cual provee como mínimo (i) un control programático de todo el control de operaciones de reenvíos, (ii) capacidad de promover su información en la red, (iii) reporte de

estadísticas, y (iv) notificaciones de eventos. Uno de los CDPI más populares es el ya mencionado OpenFlow [14]

- Interfaces hacia el norte SDN (NBI): SDN NBI son interfaces entre aplicaciones SDN y controladores SDN, los cuales brindan vistas abstractas de la red y habilitan expresiones directas del comportamiento de la red y sus requisitos. Pueden darse en cualquier nivel de abstracción y a través de distintas funcionalidades. Es habitual que estas interfaces sean del tipo REST [15].

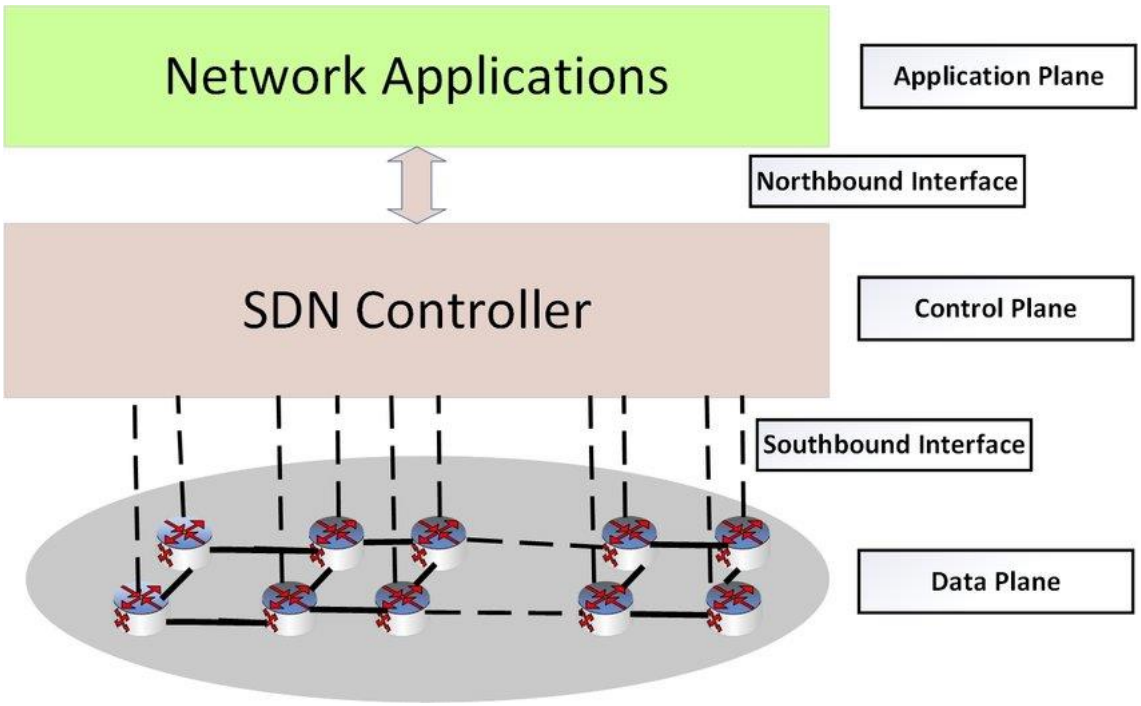

Ilustración 4 Arquitectura de SDN [I_4]

En este ejemplo la "Application Plane" sería nuestra aplicación NAC, y la interfaz Norte SDN (NBI) sería una interfaz tipo REST.

2.3.3 Razones para usar SDN

Las estructuras SDN proveen una arquitectura que transforma redes más rígidas y estáticas en sistemas dinámicos y más flexibles con una amplia variedad de posibilidades que se adecúan a las necesidades de hoy en día. Por estas razones el manejo de redes vía software va a ir (y de hecho ya está) cogiendo más y más protagonismo a medida que avance el tiempo y se siga expandiendo y desarrollando. Por otra parte, en el contexto de este proyecto, la tecnología SDN nos permite de forma unificada monitorizar y controlar de forma eficiente nuestra infraestructura de red – en particular, los switches de acceso.

2.4 OpenFlow

2.4.1 Definición

OpenFlow es una tecnología de switching que surgió a raíz del proyecto de Investigación: "OpenFlow: Enabling Innovation in Campus Networks" de 2008 en la Universidad de Stanford [14]. Es un protocolo emergente que permite a un servidor software determinar el camino de reenvío de paquetes (de ahí que se denomine tecnología de switching). Un aspecto positivo de esta tecnología es que las decisiones que impliquen el movimiento de paquetes está centralizado, por lo que la red puede ser programada sin tener que configurar individualmente cada switch, Con OpenFlow, una parte del datapath reside dentro del dispositivo, pero es un controlador el que realiza las decisiones de encaminamiento de alto nivel. Ambos elementos se comunican a través del protocolo OpenFlow.

OpenFlow permite que las redes evolucionen proporcionando un controlador que funciona mediante software, centralizado y que habilita la modificación del comportamiento de dispositivos de la red a través de un conjunto de instrucciones de reenvío. Como ya se ha comentado, en el contexto de este TFG, OpenFlow nos permite definir el comportamiento que va a tener nuestra red.

2.4.2 Estructura de OpenFlow

Los switches tradicionales utilizan STP "Spanning Tree Protocol", u otros como SPB o TRILL, para determinar cómo se reenvían los paquetes. Como hemos visto anteriormente, OpenFlow traslada esta decisión a un controlador, normalmente en un servidor o en una estación de trabajo. Las instrucciones de reenvío o encaminamiento se basan en un "flujo". Existen una infinidad de parámetros que pueden especificar un flujo como, por ejemplo, un puerto Ethernet de origen, la etiqueta VLAN, el destino Ethernet o el puerto IP. Cuando llega un paquete al switch, este busca en su tabla (o tablas) de flujos un flujo que haga "match" con las características del paquete, el cual contiene también la información de qué hacer con ese paquete.

Cuando llega un paquete y no encuentra ninguna coincidencia en las tablas de flujos, se debe crear uno nuevo. En estos casos el paquete será enviado al controlador, y este definirá un nuevo flujo para ese paquete y creará una o más entradas en las tablas de flujos. Una vez completado este proceso, el paquete será devuelto al switch y este podrá procesarlo con la tabla de flujos actualizada.

- Flujos: Cada flujo tiene asociada una acción, y a continuación describiremos tres de estas acciones básicas:
    - Reenvío del flujo de paquetes a un puerto.

    - Encapsulado y reenvío del flujo de paquetes al controlador. Este caso se da cuando llega un paquete que no coincide con ninguna entrada de la tabla de flujos y se le reenvía al controlador para que decida si añadir o no las entradas y el flujo correspondiente como mencionábamos anteriormente.
    - Descartar el flujo de paquetes.

- Tablas de Flujos: Una tabla de flujos recoge los flujos que se han ido añadiendo hasta el momento. No tiene por qué existir sólo una tabla de flujos, ya que puede haber más. Los paquetes recibidos intentarán emparejarse con las entradas de la tabla para poder saber cómo gestionar dicho paquete. Los flujos tiene los siguientes tres campos principales:
    1. La cabecera del paquete que define el flujo.
    2. La acción (denegar o permitir la circulación del paquete en cuestión).
    3. Las estadísticas. Con esta sección se puede monitorizar el número de paquetes y bytes de cada flujo y el tiempo desde el último match con el flujo (para poder depurar flujos inactivos).

- Matching: Cuando se recibe un paquete el switch OpenFlow comienza haciendo una búsqueda en la primera tabla de flujo. Cuando los campos coinciden con alguna entrada se considera un "match" y se aplica el set de instrucciones especificado por el flujo en cuestión. Los campos usados para buscar coincidencias dependen del tipo de paquete y normalmente incluyen varios campos de cabecera. Las coincidencias se pueden hacer en base al puerto de entrada, dirección de entrada o destino IP o MAC y por campos de metadatos. Otra característica importante es que los flujos tienen una prioridad asociada, y el paquete sólo hará un "match" con el flujo de mayor prioridad que coincida con las características del mismo.

- Mensajes del protocolo OpenFlow: El protocolo consiste en tres tipos de mensajes, y cada uno puede tener distintas finalidades.

    1. Del controlador al switch:
        a. Modificar o eliminar definiciones de flujos.
        b. Solicitar información de las capacidades del switch.
        c. Obtener información, por ejemplo, de los contadores del switch.
        d. Enviar paquetes de vuelta al switch después de haber añadido un nuevo flujo.
    2. Mensajes asíncronos del switch al controlador:
        a. Enviar al controlador un paquete que no coincide con los flujos existentes.
        b. Informar al controlador de la eliminación de un flujo.
        c. Informar al controlador del cambio del estado de un puerto o de un error sucedido en un switch.
    3. Mensajes simétricos (pueden enviarse desde el switch y desde el controlador):
        a. Mensajes de "hello" intercambiados al inicio.
        b. Mensajes echo para verificar la conectividad y la latencia entre el controlador y el switch.

c. Mensajes experimentales para futuras extensiones del protocolo OpenFlow.

En el contexto de este TFG, las políticas de control de acceso serán implementadas instalando flujos en los switches de la red los cuales permitirán o denegarán la comunicación entre los hosts de función de lo definido en una ACL. Esto se llevará a cabo mediante el software que hemos desarrollado, el cual tomará la información de la ACL (en nuestro caso de un fichero con un formato específico) y se comunicará mediante comandos REST con el controlador el cual instalará los flujos correspondientes en los switches especificados.

Capítulo 3. Entorno de experimentación

3.1 Herramientas utilizadas

A continuación, describiremos y analizaremos las herramientas utilizadas para desarrollar este proyecto. La utilización de Mininet y RYU fueron propuestas por mi tutor en primera instancia, pero el uso de Python fue opcional ya que también se podía haber desarrollado mediante otro lenguaje de programación. Esto se debe a que utilizamos interfaces norte REST y existen APIS para ello en otros lenguajes de programación.

3.2 Mininet

3.2.1 Entornos Virtuales

Se considera software de entorno virtual cualquier tipo de programa o sistema que implemente, gestione o controle múltiples instancias de entornos virtuales. Es decir, un software que simule un sistema de computación y pueda ejecutar programas como si fuese un ordenador o sistema real.

Usar entornos virtuales es muy beneficioso ya que reduce costes, por ejemplo, de cara a este proyecto no necesito comprar y montar una red local, sino que puedo crearla con las características que necesite y con un comportamiento real. Además es positivo de cara a crear copias de seguridad de sistemas enteros y poder recuperarlos en caso de que surja algún fallo o avería, entre otras muchas ventajas.

3.2.2 ¿Qué es Mininet?

Mininet es un emulador de red que crea un grupo de hosts, switches, routers y enlaces virtuales dentro de un sistema Linux que hace de anfitrión. Mininet proporciona una base para el desarrollo de SDN creando redes funcionales con las características necesarias. Las redes de Mininet ejecutan código real incluyendo aplicaciones Unix/Linux estándar como aplicaciones web. Además, cuenta con una amplia API de Python dedicada a la creación y experimentación de diversas redes. Los switches de

Mininet (Open vSwitch [13]) son compatibles con OpenFlow, y esa es una de las razones más importantes para utilizar Mininet de cara a este proyecto.

Mininet utiliza virtualización basada en procesos para ejecutar los hosts y switches. El máximo número de hosts ejecutados simultáneamente han sido 4096 según destaca la página principal de Mininet [8], por lo tanto, nos podemos hacer una idea de las dimensiones de las redes que podemos llegar a crear de cara a su estudio, experimentación etc.

Mininet es una herramienta rápida, de instalación sencilla con muy buena escalabilidad y además del resto de puntos fuertes señalados hasta el momento, también puede conectarse a redes reales. No obstante, como toda herramienta también cuenta con limitaciones. En primer lugar, las redes basadas en Mininet no pueden superar el CPU o el ancho de banda habilitado en un servidor. Y por último no puede ejecutar aplicaciones o switches no compatibles con Linux, aunque no ha llegado a ser un inconveniente en la práctica.

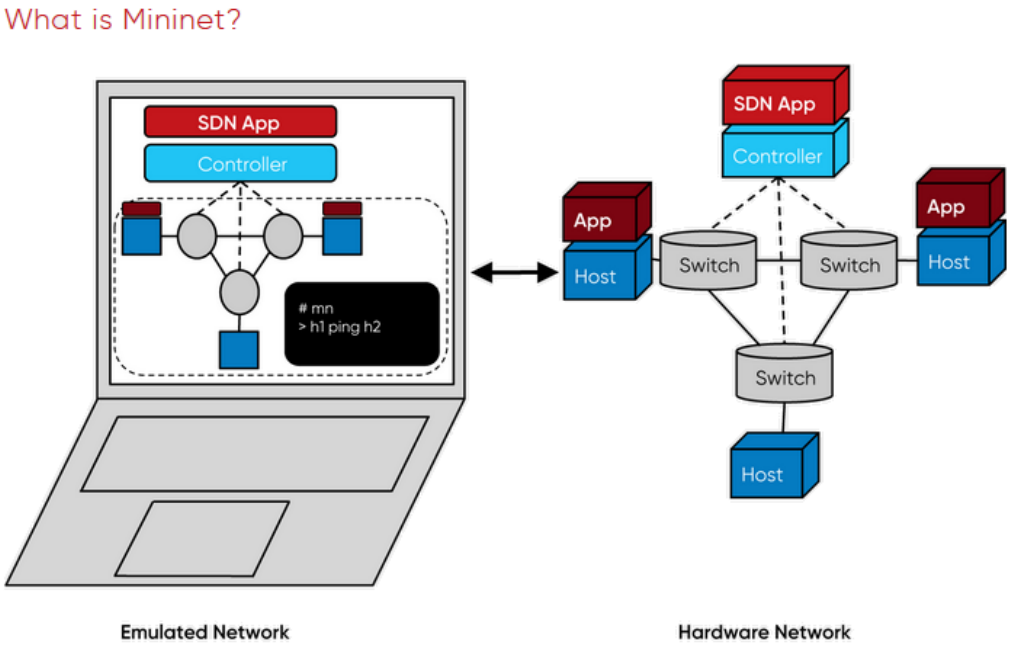

Ilustración 5 Esquema Mininet [I_5]

3.3 RYU

RYU es un framework de SDN-OpenFlow basado en componentes. RYU proporciona software, controlador y APIs bien definidas que facilitan la creación de aplicaciones de gestión de red y de control. La interfaz sur es OpenFlow, y se incluyen facilidades para interactuar con las aplicaciones de control (interfaz norte) mediante funciones Python y mediante interfaces REST.

La documentación de RYU [10] y de sus módulos es bastante completa y es aquí donde se puede empezar a aprender cómo utilizarlo y a familiarizarte con la misma. Un punto positivo es que parten de ejemplos reales como en el capítulo de “Switching Hub” para enseñarte a programar aplicaciones SDN utilizando RYU.

En este TFG se usa el controlador RYU conjuntamente con dos aplicaciones de control ya implementadas que forman parte del paquete de instalación: Rest_Firewall y ofctl_rest. Estas aplicaciones permiten actuar sobre la infraestructura y monitorizarla, mediante una interfaz REST.

3.3.1 Rest_Firewall

Rest_Firewall es una aplicación RYU orientada a la implementación de cortafuegos, que incorpora una serie de funcionalidades que nos han sido muy útiles a la hora de implementar nuestro sistema NAC. Mediante la interfaz REST que ofrece, nos permite añadir reglas que habiliten o denieguen el paso de paquetes basándonos en dirección de origen y destino (ya sea MAC, IPV4, IPV6 o puerto de entrada).

La forma en la que contactamos desde el exterior (ya sea desde nuestra aplicación NAC o desde un terminal) es mediante peticiones HTTP. Dichas peticiones se pueden realizar mediante funciones CURL [16]. Podemos habilitar o deshabilitar un switch como filtro de paquetes, leer el estado de los switches de la red (para ver si están habilitados o no), añadir o eliminar reglas y obtener el listado de reglas activas.

La razón por la que escogimos Rest_Firewall fue por su versatilidad y porque nos proporcionaba una forma de llevar a cabo el NAC, ya que, una vez diseñado una estructura para el ACL, podemos crear reglas y traducirlas al comando curl (petición HTTP) apropiado para que sean aplicadas en la red. Por poner un ejemplo de la propia documentación de RYU, fijémonos en la red de la Ilustración 6.

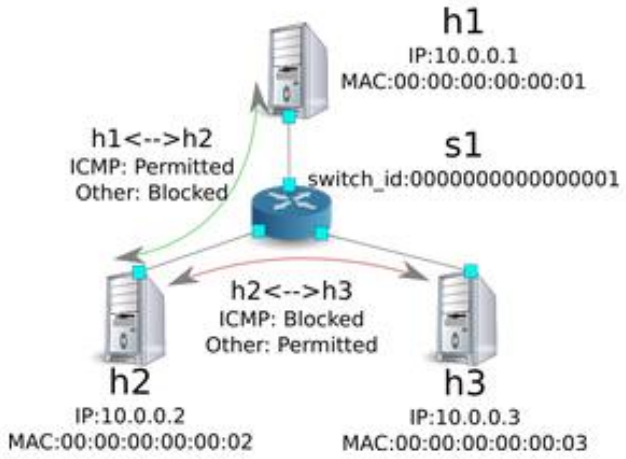

Ilustración 6 Ejemplo Rest_Firewall [I_6]

En este caso queremos permitir la comunicación ICMP y bloquear el resto desde el host 1 al host 2. En Rest_Firewall, por defecto todas las comunicaciones están bloqueadas, así que simplemente tendremos que habilitar ICMP de h1 a h2 (¡y viceversa!). En el capítulo 2.2.3 comentábamos por qué se daba esto, y es que dependiendo de la política de acceso que queramos, la última regla a la cual se llegará cuando el paquete no ha sido emparejado por ninguna regla previa se encargará de denegar el acceso al paquete o de permitírselo. En el ejemplo que estamos analizando, las reglas se utilizarán para permitir el paso de los paquetes y la última regla se encarga de denegar el resto (el flow correspondiente a esto último se instala automáticamente cuando iniciamos Rest_Firewall). Por lo tanto, los dos primeros comandos a ejecutar serán los siguientes:

1) curl -X POST -d '{"nw_src": "10.0.0.1/32", "nw_dst": "10.0.0.2/32", nw_proto": "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001
2) curl -X POST -d '{"nw_src": "10.0.0.2/32", "nw_dst": "10.0.0.1/32", nw_proto": "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001

De esta forma hemos logrado la comunicación ICMP del h1 al h2, ahora toca hacer lo contrario entre h2 y h3. Es importante recordar que en un ACL una vez se dé el primer match se deja de buscar, por lo tanto, debemos escribir la regla más restrictiva primero (bloquear ICMP) primero, o con más prioridad, y luego la regla en la que habilitamos cualquier comunicación:

1) curl -X POST -d '{"nw_src": "10.0.0.2/32", "nw_dst": "10.0.0.3/32", "nw_proto": "ICMP", "actions": "DENY", "priority": "10"}' http://localhost:8080/firewall/rules/0000000000000001
2) curl -X POST -d '{"nw_src": "10.0.0.3/32", "nw_dst": "10.0.0.2/32", "nw_proto": "ICMP", "actions": "DENY", "priority": "10"}' http://localhost:8080/firewall/rules/0000000000000001
3) curl -X POST -d '{"nw_src": "10.0.0.2/32", "nw_dst": "10.0.0.3/32"}' http://localhost:8080/firewall/rules/0000000000000001
4) curl -X POST -d '{"nw_src": "10.0.0.3/32", "nw_dst": "10.0.0.2/32"}' http://localhost:8080/firewall/rules/0000000000000001

De esta forma obtenemos las reglas apropiadas para formar esta red simple. Y esto se ve reflejado en los flujos de la red. Podemos comprobarlo accediendo al switch, en este caso el único que tenemos accediendo a la consola de Mininet y escribiendo el comando “xterm s1”, y una vez dentro usaremos “ovs-ofctl -O openflow13 dump-flows s1”. El resultado es el reflejado en la ilustración 7.

```
# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x3, duration=242.155s, table=0, n_packets=0, n_bytes=0, priority=1,ip,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=NORMAL
 cookie=0x4, duration=233.099s, table=0, n_packets=0, n_bytes=0, priority=1,ip,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x0, duration=1270.233s, table=0, n_packets=10, n_bytes=420, priority=65534,arp actions=NORMAL
 cookie=0x0, duration=989s, table=0, n_packets=20, n_bytes=1960, priority=0 actions=CONTROLLER:128
 cookie=0x5, duration=26.984s, table=0, n_packets=0, n_bytes=0, priority=10,icmp,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=CONTROLLER:128
 cookie=0x1, duration=591.578s, table=0, n_packets=0, n_bytes=0, priority=1,icmp,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x6, duration=14.523s, table=0, n_packets=0, n_bytes=0, priority=10,icmp,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=CONTROLLER:128
 cookie=0x2, duration=564.793s, table=0, n_packets=0, n_bytes=0, priority=1,icmp,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=NORMAL
```

Ilustración 7 dump-flows en S1 del ejemplo

El tercer y cuarto Flow vienen por defecto una vez inicias el Rest_Firewall y su función es habilitar ARP (Address Resolution Protocol) y denegar los paquetes que no hagan match con el resto de reglas respectivamente (el flow que mencionábamos antes, es por eso que tiene prioridad 0). Los dos primeros flows son los que permiten la comunicación entre h2 y h3. Además, el quinto y séptimo flow representan el bloquea a las tramas ICMP entre h2 y h3. Por último, el sexto y octavo flow habilitan el ICMP entre h1 y h2. En este caso no aparecen ordenadas, pero es la prioridad la que establece el orden correcto, aún que en nuestra implementación la prioridad será establecida de forma ordenada y decreciente de forma que la primera regla tenga la prioridad más alta y vaya decreciendo con el paso del resto de reglas.

Para concluir con Rest_Firewall adjunto en la Ilustración 8 los argumentos y la serie de valores que pueden aceptar los mismos dentro de las funciones a las que se accede vía REST para poder crear una regla. Esta información va a ser relevante a la hora de diseñar y crear nuestro ACL.

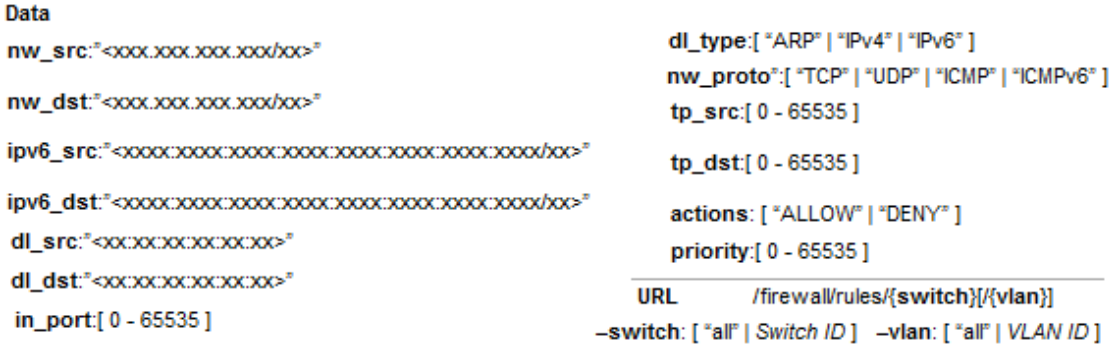

Ilustración 8 Campos para las reglas de Rest_Firewall

3.3.2 Ofctl_Rest

Ofctl_rest [12] es una aplicación RYU que ofrece una API REST con una amplia variedad de opciones para interrogar switches y conseguir información sobre ellos o incluso modificar esta información. Además de esto se puede utilizar para depurar aplicaciónes y conseguir estadísticas de la red.

En el contexto del TFG utilizamos ofctl_rest junto al Rest_Firewall para poder interrogar a la red y saber cuántos switches hay, y una vez sabemos la cantidad, podemos interrogarlos uno a uno para conseguir la información que nos da el comando "dump-flows" visto previamente, pero en nuestro programa principal (en lenguaje Python). En el apartado de la implementación veremos por qué es importante, pero de momento vamos a mirar los dos comandos que vamos a utilizar:

```
# REST API
for switch
configuration
```

```
#
# Retrieve the switch stats
#
#
# get the list of all switches
# GET /stats/switches
#
# get the desc stats of the switch
# GET /stats/desc/<dpid>
#
# get flows desc stats of the switch
# GET /stats/flowdesc/<dpid>
```

Como podemos observar, el primer comando que mencionábamos era el obtener la lista de switches (curl -X GET http://localhost:8080/stats/switches), el cual nos devuelve una lista con los switches de la red (p. ej. [1, 2, 3] en una red con 3 switches). Es importante destacar que estos números, en este ejemplo del 1 al 3, son el "dpid" de los switches de la red, es decir, un identificador único de un switch . Por otro lado, tenemos el comando curl que nos proporciona la misma información que cuando accedemos a un switch desde Mininet y usamos el comando "dump-flows" (curl -X GET http://localhost:8080/stats/flow/{dpid}). De esta forma nuestra aplicación NAC podrá acceder a esta información para conseguir así una de las funcionalidades deseadas que comentaremos en el capítulo 4.

3.4 Python

3.4.1 Pycurl

Utilizamos Pycurl para interactuar con las aplicaciones de control solicitando la ejecución de acciones y analizando los resultados que nos devuelven. Es una interfaz de libcurl que nos permite realizar desde dentro de nuestra aplicación NAC en Python todos los comandos curl mencionados hasta ahora. El principal propósito y uso para pycurl es automatizar transferencias de archivos o secuencias de operaciones. A continuación, vamos a ver un ejemplo:

- curl -X GET http://localhost:8080/stats/flow/1

Este comando es usado para interrogar al switch con dpid 1, y obtener así la información de la tabla de flujos de dicho switch. En el software desarrollado utilizamos este comando para poder crear el fichero con el formato de nuestra ACL partiendo de los flujos existentes. Pero para poder utilizar este comando, necesitamos la librería Pycurl, ya que el comando que acabamos de ver se puede ejecutar en un terminal, pero nosotros necesitamos utilizarlo en nuestra aplicación NAC. A continuación, veremos, como ejemplo, cómo traducimos ese comando al programa, ya que la estructura es muy similar para cada acción (GET, PUT, POST, DELETE):

```
b_obj = BytesIO()
crl.setopt(crl.URL, 'http://localhost:8080/stats/flow/1’)
crl.setopt(crl.WRITEDATA, b_obj)
crl.perform()
get_body = b_obj.getvalue()
info_str = get_body.decode('utf8')
```

En el primer “setopt” añadimos la URL a la que deseamos acceder, y en el segundo cuando utilizamos “crl.WRITEDATA” estamos especificando que deseamos un comando GET, es por eso que en el segundo argumento dejamos una variable Bytes la cual decodificamos para obtener la respuesta del comando curl. Como hemos dicho antes, este es un comando “GET”, así que lo lanzamos con “crl.perform()” y luego debemos recoger la información que nos devuelvan y decodificarla con las dos siguientes líneas de código para poder obtener el resultado en formato string.

A continuación, vamos a ver otro ejemplo, pero esta vez ejecutaremos un comando POST. Como el nombre indica, el objetivo será transferir datos al url deseado, por lo tanto, el primer comando “setopt” será igual que en el ejemplo anterior, es decir especificaremos la URL con la que queremos interactuar. A continuación, prepararemos en la variable “data” la información que queremos enviar, y lo haremos utilizando un diccionario (una colección que se utiliza para almacenar valores por parejas “key:value”). Para finalizar convertimos el diccionario en formato json para poder enviarlo y utilizaremos “perform()” de nuevo para ejecutar el comando:

```
crl.setopt(crl.URL, 'http://localhost:8080/firewall/rules/0000000000000001)
data["priority"] = "10” | o de la forma siguiente: data = {“priority” : “10”}
pf = json.dumps(data)
çrl.setopt(crl.POSTFIELDS, pf)
crl.perform()
```

3.4.1 Requests

Requests es una librería de Python que permite llevar a cabo HTTP requests como indica su nombre. Esta librería deja a un lado las complicaciones de trabajar con HTTP/1.1 en Python, y habilita una forma más sencilla e intuitiva de llevar a cabo tareas simples como POST o DELETE entre otros. Como acabamos de ver, utilizamos Pycurl para estas situaciones en nuestra aplicación NAC, pero en el caso de DELETE, Pycurl tiene una limitación la cual no le permite añadir parámetros de la misma forma que lo hacemos, por ejemplo, en un POST. Esto es un problema ya que necesitamos añadir un parámetro para el “rule_id” en el caso de utilizar el DELETE para utilizar una de las funcionalidades de nuestro controlador, que nos permite eliminar una o más reglas vigentes en la red. Esta es la razón principal por la que utilizamos request, ya que esta librería sí que nos permite añadir dicho parámetro, y veremos un ejemplo a continuación para comprender mejor este funcionamiento.

```
mydata = {}
mydata['rule_id'] = "4"
URL_delete = "http://localhost:8080/firewall/rules/all/all"
r = requests.delete(URL_delete, data=json.dumps(mydata))
```

Comenzamos el ejemplo creando el diccionario mydata, al cual añadimos la pareja (“key:value”) de: {“rule_id” : “4”}. Este es el parámetro que necesitamos enviar para poder eliminar la regla con el id 4, pero podemos escribir “all” en su lugar para eliminar todas las reglas. A continuación, creamos la URL, para poder comunicarnos con el controlador y hacerle llegar la petición de DELETE. En este caso los dos últimos apartados de la url (“all/all”) son relevantes ya que el primero es el lugar donde podemos especificar la id del switch, y el siguiente la id de la vlan a la que queremos mandar la petición de eliminar una regla. En este caso ambas son “all” por lo tanto se eliminaría la regla 4 de todas las tablas de flujos de todos los switches.

Capítulo 4. Implementación

4.1 Implementación del sistema NAC

4.1.1 Estructura de las ACL

A continuación, vamos a definir la estructura de las ACLs que controlan nuestro NAC. Es importante tener presente la Ilustración 9 la cual nos mostraba todas las características que puede tener en cuenta Rest_Firewall a la hora de implementar una regla, así que tenemos que cubrir todas las posibilidades.

```
NOVLANID/VLANID:xx..
ALL/SWID:xx...

[MAC/PORT/IPV4/IPV6]  {src/- }  {dst/-} {dl_type/-} {nw_proto/-} {tp src} {tp dest} {actions/-}  {Bidirectional}
(PORT = in_port)
```

Ilustración 9 Estructura ACL

Comenzamos con el primer bloque en el cual especificamos si hay VLAN o no, en el caso de que vayamos a trabajar con VLAN es importante mantener la estructura de "VLANID:12" por ejemplo, si la ID de la VLAN a la que queremos añadir reglas es la 12. Lo mismo pasa si queremos que las reglas que vienen a continuación sean para un switch en concreto o para todos. Este primer bloque (VLAN y switch) determina a dónde se dirigen las reglas siguientes, y en el caso de que queramos otras reglas para otro switch o VLAN en concreto simplemente volvemos a escribir ese bloque (p. ej. NOVLAN y SWID:0000000000000002 para escribir a continuación las reglas que queremos para el segundo switch). En el caso de que queramos las reglas para todos los switches podemos escribir "ALL" en la sección de SWID para que las reglas sean destinadas a todos los switches de la red.

A continuación, llegan las reglas, organizadas en filas de 9 elementos. Cuando queramos especificar el elemento lo escribiremos respetando el formato de la Ilustración 9, y cuando no necesitemos hacerlo escribiremos "-" en ese hueco:

- Comenzamos con el primer elemento que nos da la información del tipo de identificación de procedencia y destino (src y dst), y hay 4 opciones: MAC, IPV4, IPV6 y PORT. El segundo y tercer elemento tendrán que corresponderse con el protocolo escogido en la primera opción. En el caso de PORT, el apartado de procedencia debe de ser un puerto ("in_port" = [0-65535] y el destino, si lo hay, puede ser una dirección MAC, IPV4 o IPV6. EN EL CASO DE DIRECCIONES IPV4/IPV6, LOS DESTINOS/ORÍGENES PUEDEN SER

MÁS DE UN NODO, EN FUNCIÓN DE LA MÁSCARA. SOLO SI LA MÁSCARA ES /32 LA IDENTIFICACIÓN ES DE UN NODO.

- A continuación, tenemos el dl_type (que sólo pueden ser: ARP o IPV4/6).
- Seguido se encuentra nw_proto (que puede ser: TCP, UDP, ICMP, ICMPV6).
- Las siguen el puerto origen y destino correspondientes al protocolo de transporte elegido (TCP o UDP), que tiene que estar dentro del rango [0-65535].
- Para finalizar también escogemos la acción (ALLOW/DENY, por defecto es ALLOW).
- También podemos activar la opción de bidireccional "1", la cual nos ahorra espacio en el caso de que queramos aplicar la misma regla de esta línea, pero con el segundo y tercer elemento cambiados (src y dst). Esto es útil ya que muchas veces necesitamos hacer la misma regla en dos direcciones, como en el ejemplo del cortafuegos.

Es importante recordar que la prioridad no será un campo a rellenar ya que se irá asignando la mayor prioridad a la primera regla y se irá decrementando por una unidad y asignando a la siguiente regla. Es así como se asegura que el orden tenga la relevancia que debe de tener en una ACL.

A continuación, se presenta un ejemplo de fichero de configuración con dicho formato, que configuraría la red de la ilustración 6. No hace falta que los elementos estén alineados, con que estén separados mínimo por un espacio es suficiente, pero de esta forma es más fácil leerlo:

```
NOVLAN
ALL

IPV4  10.0.0.1/32  10.0.0.2/32  -  ICMP  -  -  -      1
IPV4  10.0.0.2/32  10.0.0.3/32  -  ICMP  -  -  DENY   1
IPV4  10.0.0.2/32  10.0.0.3/32  -  -     -  -  -      1
```

Para este ejemplo se ha utilizado la dirección IP utilizando una máscara de 32 para identificar cada nodo, pero se podría haber utilizado sus direcciones MAC o incluso los números de los puertos del switch al que los nodos están conectados. Por último, recordar también que como ya hemos visto previamente, no es necesario añadir de forma explícita en el fichero una regla "catch all" denegando el resto de los paquetes que no coincidan con estas reglas, ya que Rest_Firewall lo hace por defecto. En el caso de que la política de acceso sea la contraria sí que debemos de indicarlo explícitamente y eliminar la regla que viene añadida por defecto (y veremos en breve el comando que utilizaremos para poder eliminar cualquier regla que deseemos).

4.1.2 Configuración desde un fichero

A continuación, en la Ilustración 10, tenemos el pseudo código de la implementación de una de las funcionalidades clave de nuestra aplicación NAC. Consiste en leer un fichero de configuración como el que hemos visto previamente y

convertir las reglas escritas en dicho fichero en comandos CURL para comunicarnos con el controlador de la red y que pueda añadir los flows adecuados.

Esta funcionalidad se divide en dos fases: fase de comprobación y fase de ejecución. ¿Se podrían hacer ambas a la vez? La respuesta es sí, pero si se hace a la vez y hay un problema en, por ejemplo, la tercera regla, el programa parará, pero las dos primeras ya habrán sido instaladas en los switches. Por lo tanto, decidimos separarlas y que sólo se ejecuten las reglas una vez se haya verificado el fichero entero.

```
1) Fase de comprobación

    Leemos el fichero, por cada línea:

    IF 1 palabra en la línea: #Significa que es la parte de vlan y switchid

        Comprobar que el formato sea correcto

    ELSE: #Significa que es la parte de las normas

        Verificar que el SRC y DST encajen con el formato del primer elemento

        comprobar que el dl_tyoe y el nw_proto sean sus respectivas opciones posibles

        Comprobar que no haya tp_dst y tp_src en el caso de que el protocolo sea ICMP o ICMPv6

        Comprobar que tpsrc y tpdst estén dentro del rango [0-65535]

        Comprobar que la acción sea ALLOW/DENY/-

        (si hay algún error se notifica por texto y se activa una variable Flag = 1)

IF Flag = 1

    Hay algún fallo de formato, no se ejecutan las normas

Else

2) Fase de ejecución

    Leemos el fichero, por cada línea:

    IF 1 palabra en la línea: #Significa que es la parte de vlan y switchid

        preparar la vlan y la swid marcado

    ELSE: #Significa que es la parte de las normas

        añadir al diccionario cada elemento que no sea "-"

        ejecutar la orden curl

        if Bidireccional == "1":

            Ejecutarlo de nuevo con el src y dst cambiados
```

Ilustración 10 Pseudocódigo de la funcionalidad

A continuación, en la ilustración 11, se muestran tres funciones las cuales comprueban que el string que se les dé cumple el formato propio de una dirección MAC, IPV4 o IPV6. Esto es especialmente útil a la hora de realizar la comprobación del fichero con la estructura y el formato de la ACL, para asegurar que si se especifica un protocolo, la dirección escrita coincida con la estructura que debería.

Por otro lado, en la ilustración 12 mostramos una sección de la parte en la que comprobamos que los dos primeros atributos coinciden con el formato escogido en el atributo 0 (MAC, IPV4/6, IN_PORT).

```
#Cheks if the ipv4 address is correct (with mask)
def check_ipv4(ip):
    #mask
    str = ip.split("/")
    if len(str) != 2:
        return False

    i = int(str[1])
    if i < 0 or i > 32:
        return False
    #ip
    try:
        socket.inet_aton(str[0])
        return True
    except socket.error:
        return False
```

```
#Cheks if the ipv6 address is correct
def check_ipv6(ip):
    try:
        socket.inet_pton(socket.AF_INET6, ip)
        return True
    except socket.error:
        return False

def check_mac(mac):

    if re.match("[0-9a-f]{2}([-:]?)[0-9a-f]{2}(\\1[0-9a-f]{2}){4}$", mac.lower()):
        return True
    else:
        return False
```

Ilustración 11 Listado de código 1

```
#First and second attribute SRC
if t == 0:
    if not(check_mac(attributes[1])):
        print("Syntax error, MAC structure expected in Source field at line: {}\n".format(linec))
        flag = 1
    if attributes[2] != "-": #!! Double if because F and F = T but we only want it to enter if both are T
        if not(check_mac(attributes[2])):
            print("Syntax error, MAC structure expected in Destination field at line: {}\n".format(linec))
            flag = 1

if t == 3:
    if not(check_ipv4(attributes[1])):
        print("Syntax error, IPV4 structure expected (with mask) in Source field at line: {}\n".format(linec))
        flag = 1
    if attributes[2] != "-":
        if not(check_ipv4(attributes[2])):
            print("Syntax error, IPV4 structure expected (with mask) in Destination field at line: {}\n".format(linec))
            flag = 1

if t == 5:
    if not(check_ipv6(attributes[1])):
        print("Syntax error, IPV6 structure expected in Source field at line: {}\n".format(linec))
        flag = 1
    if attributes[2] != "-":
        if not(check_ipv6(attributes[2])):
            print("Syntax error, IPV6 structure expected in Destination field at line: {}\n".format(linec))
            flag = 1

if t == 2:
    try:
        i = int(attributes[1])
    except:
        print("Syntax error at line: {}, in_port must be in range [0-65535] and must be a number \n".format(linec))
        flag = 1
        break
    if i < 0 or i > 65535 :
        print("Syntax error at line: {}, in_port must be in range [0-65535] \n".format(linec))
        flag = 1

    if attributes[2] != "-":
        if not(check_mac(attributes[2]) or check_ipv4(attributes[2]) or check_ipv6(attributes[2])):
            print("Syntax error, MAC, IPV4 or IPV6 structure expected in Destination field at line: {}\n".format(linec))
            flag = 1
```

Ilustración 12 Listado de código 2

Lo más destacado de lo mostrado en el listado de código 2 es que, si se especifica que el primer elemento es “PORT”, el siguiente elemento (origen) debe de ser un puerto, pero el siguiente (destino) puede ser tanto una dirección MAC o IPV4/6, y esto se tiene en cuenta a la hora de hacer la comprobación correcta (este es el caso t == 2). Hay otras comprobaciones importantes como por ejemplo el que no pueda haber tp_src y tp_dst si el protocolo no es ni TCP ni UDP. Seguido a la verificación, si no ha habido ningún contratiempo, pasaríamos a la fase de ejecución, en la cual sabemos que todo está en orden por lo tanto las zonas en las que sólo hay una palabra son para preparar el destino (con el swid y el vlanid) y el resto son las zonas donde se encuentran las reglas y se puede ir añadiendo cada elemento que no sea “-“ a un diccionario llamado “data” que luego es enviado con el comando curl para añadir el flow correspondiente al controlador.

Por otro lado, destacar que una vez se comprueba que el formato del fichero es correcto, antes de proceder a aplicar las reglas del mismo, se le preguntará al usuario si desea eliminar las reglas vigentes (es decir, empezar de 0 y añadir las reglas nuevas posteriormente) o no. Creemos que esto es relevante ya que a pesar de que el usuario dispone del comando “delete” para eliminar las reglas que desee (incluso hacer una limpieza de todas las reglas) y el comando rules para saber en todo momento las reglas vigentes, suele ser habitual cargar reglas de un fichero partiendo de 0, por lo tanto, si así lo desea el usuario se eliminarán todas las reglas, y de no ser así se añadirán las reglas del fichero a las reglas actuales. Debemos tener en cuenta que como hemos explicado las reglas se van añadiendo con una prioridad decreciente, por lo tanto, las reglas que añadamos primero tendrán más pero que las anteriores para darle relevancia al orden, y esta es otra razón por la cual preguntar al usuario si desea eliminar las reglas vigentes es importante.

```
for line in file:

    for word in line.split():
        attributes[count] = word
        count += 1

    if count == 1: #change switch ID or VLAN
        if word == "NOVLAN":
            vlan = " "

        elif "VLANID:" in word:
            id = word.split(":")
            vlan = ("/" + id[1])

        elif word == "ALL":
            swid = "all"

        elif "SWID:" in word:
            id = word.split(":")
            swid = id[1]

        else:
            print("Syntax ERROR at line {}".format(linec))
            break

    crl.setopt(crl.URL, 'http://localhost:8080/firewall/rules/{}{}'.format(swid,vlan))
```

Ilustración 13 Listado de código 3

```
#Forth Attribute (nw proto)
if attributes[4] != "-":
    data["nw_proto"] = "{}".format(attributes[4])

#Fifth Attribute (tp_src)
if attributes[5] != "-":
    data["tp_src"] = "{}".format(attributes[5])

#Sixth Attribute (tp_dst)
if attributes[6] != "-":
    data["tp_dst"] = "{}".format(attributes[6])

#Seventh Attribute (actions)
if attributes[7] != "-":
    data["actions"] = "{}".format(attributes[7])

#Priority Automatically added
data["priority"] = "{}".format(priority)
priority -= 1


#print("DATA: {}\n\n".format(data))
pf = json.dumps(data)
crl.setopt(crl.POSTFIELDS, pf)
crl.perform()

if attributes[8] == "1": #Bidirectional
    data["{}".format(types[t])] = "{}".format(attributes[2])
    data["{}".format(types[t+1])] = "{}".format(attributes[1])
    data["priority"] = "{}".format(priority)
    priority -= 1
    #print("DATA: {}".format(data))
    pf = json.dumps(data)
    crl.setopt(crl.POSTFIELDS, pf)
    crl.perform()
```

Ilustración 14 Listado de código 4

4.1.3 Proceso inverso

A continuación, veremos el seudocódigo y parte de la implementación del proceso inverso al que hemos visto previamente, es decir, vamos a leer los flows y crear un archivo de configuración con el formato del ACL de las reglas establecidas hasta el momento en la red. Esta funcionalidad es útil para guardar copias de seguridad de las reglas actuales, y posteriormente pueden ser implementadas de nuevo utilizando la funcionalidad estándar de crear reglas a raíz de un archivo.

```
Comando CURL para obtener la lista de dpid de los switches

Crear un fichero de texto vacío

For s in Lista_Switches:

    output = Comando curl para obtener "dump flows" del switch s

    Dividir el texto del output por secciones de cada flows

    Para cada sección del flow: # Buscaremos los 8 elementos: MAC/IPV4/6/PORT | src | dst | dl_type
                                                              | nw_proto | tp_src | tp_dst | action

        Comprobar si pertenece a una VLAN para completar el bloque SWID y VLAN

        Buscar "dl_src", "nw_src", "ipv6_src" o "in_port" en la sección
            (de aquí obtenemos los tres primeros elementos, comprobando si tiene dst)

        Buscar "dl_type" en el fragmento

        Buscar "nw_proto" en el fragmento

        Buscar "tp_src" y "tp_dst"

        Mirar el Action

        Crear la norma y escribirla en el fichero
```

Ilustración 15 Pseudocódigo

Estos son los pasos seguidos en esta implementación:

- Utilizamos el comando curl -X GET http://localhost:8080/stats/switches, el cual devuelve una lista de dpid de los switches de la red. Este comando se traduce así usando Pycurl:

```
b_obj = BytesIO()
crl = pycurl.Curl()
crl.setopt(crl.URL, 'http://localhost:8080/stats/switches')
crl.setopt(crl.WRITEDATA, b_obj)
crl.perform()
get_body = b_obj.getvalue()
info_str =  get_body.decode('utf8')
```

Ilustración 16 Listado de código 5

- A continuación, abrimos el fichero en el que iremos escribiendo las reglas y comenzaremos a iterar sobre cada switch, del cual conseguiremos la misma información que usando el comando de “dump-flows” desde Mininet, pero al ser desde la aplicación NAC lo haremos mediante REST (con este comando: curl -X GET http://localhost:8080/stats/flow/{dpid}):

- Dicho comando devuelve el texto con todos los flows, pero dividiremos ese output de tal forma que en cada porción del texto quede un flow, y el siguiente paso se realizará iterando sobre cada porción (es decir sobre cada flow).

- Comprobaremos que el flow pertenece a la misma VLAN (si tiene) que el anterior, y de no ser así crearemos otro "bloque" con el SWID actual y el VLAN correspondiente. Tras este paso, comprobaremos las 4 opciones que pueden formar el src, y de ahí también conseguiremos la información del primer elemento (MAC/IPV4/6/PORT). Todo esto se irá almacenando en un array el cual está inicializado a "-", por lo tanto el elemento que no encontremos permanecerá de esta forma para ser compatible con el formato de nuestro ACL. El código de esta parte es el siguiente (recordar que el dst de un "in_port" sólo puede ser una dirección MAC, IPV4 o IPV6):

```
#src
atr = ["-"]*8
if "dl_src" in l:
    atr[0] = "MAC"
    atr[1] = get_next("dl_src",l)

elif "nw_src" in l:
    atr[0] = "IPV4"
    src = get_next("nw_src",l)
    if "/" in src:
        atr[1] = src
    else:
        atr[1] = src + "/32"

elif "ipv6_src" in l:
    atr[0]= "IPV6"
    atr[1] = get_next("ipv6_src",l)

elif "in_port" in l:
    atr[0]= "PORT"
    atr[1] = get_next("in_port",l)
#dst
if "dl_dst" in l:
    atr[2] = get_next("dl_dst",l)

elif "nw_dst" in l:
    dst = get_next("nw_dst",l)
    if "/" in src:
        atr[2] = dst
    else:
        atr[2] = dst + "/32"

elif "ipv6_dst" in l:
    atr[2] = get_next("ipv6_dst",l)
```

Ilustración 17 Listado de código 6

- A continuación, tenemos la forma de buscar los siguientes elementos. Señalar que tanto get_next() como get_next2() son funciones que devuelven la parte de texto que contiene la respuesta necesaria, y como se repite siempre el mismo patrón, estas funciones fueron creadas para para ahorrar la repetición de código. En el caso de dl_type, nw_proto y la acción, debemos fijarnos en ciertos números que aparecen en el flow para poder identificar el protocolo o la acción que es. Por último, se escribe en el archivo la información recopilada que hayamos encontrado en dicho flow. La única parte negativa es que no usaremos la parte de bidireccional en esta implementación, pero es una opción que se puede utilizar al leer el archivo sin problema.

```
#dl_type
if "dl_type" in l:
    aux = get_next2("dl_type",l)
    if aux == "2048":
        atr[3] = "IPv4"
    elif aux == "34525":
        atr[3] = "IPv6"
    else: #2054
        atr[3] = "ARP"

#nw_proto
if "nw_proto" in l:
    aux = get_next2("nw_proto",l)
    if aux == "1":
        atr[4] = "ICMP"
    elif aux == "58":
        atr[4] = "ICMPv6"
    elif aux == "17":
        atr[4] = "UDP"
    else:
        atr[4] = "TCP"

#tp_src
if "tp_src" in l:
    atr[5] = get_next("tp_src",l)

#tp_dst
if "tp_dst" in l:
    atr[6] = get_next("tp_dst",l)

#Actions : Allow/normal -> OUTPUT:4294967290 || Deny/controller -> OUTPUT:4
if "OUTPUT:4294967293" in l:
    atr[7] = "DENY"
else:
    atr[7] = "ALLOW"

if atr[0] != "-":
    f.write("{}    {}    {}    {}    {}    {}    {}    {}    -\n".format(atr[0],
            atr[1], atr[2], atr[3], atr[4], atr[5], atr[6], atr[7],))
```

Ilustración 18 Listado de código 7

De esta forma conseguimos la funcionalidad inversa a la anterior. Como acabamos de mencionar, no usamos el apartado de bidireccional por lo tanto la ACL quedará un poco más larga, ya que en el ejemplo mostrado lo usábamos en todas las reglas, y en el caso de no usarlo tendremos el doble de reglas, una desde el destino A al B y viceversa, que es precisamente lo que nos ahorra este atributo de la ACL. Por otro lado, los flujos quedan ordenados una vez se añaden por que se van añadiendo con una prioridad decreciente, y ese orden es el mismo que conseguiremos en la ACL que crearemos con la versión inversa también. Por último, aclarar también que cada flujo de la tabla corresponde a una regla y viceversa, ya que cada regla en la ACL crea dicho flujo.

4.1.4 Otras funcionalidades

La aplicación NAC que se ha diseñado tiene el formato de una consola, es decir, una vez sea ejecutado podremos escribir en ella los distintos comandos que tenemos disponibles. Hay varias funcionalidades desde QoL “Quality of Life”, es decir, funcionalidades que hacen más sencillo el uso del programa, hasta comandos relacionados con Rest_Firewall.

A continuación, tenemos el comando “commands” el cual nos imprime una lista de los comandos disponibles hasta el momento y una breve descripción. Los dos primeros se utilizan para abrir de forma rápida un terminal con Mininet con una red de prueba, y otro terminal con Rest_Firewall y ofctl_rest. Openex2 hace lo mismo pero ejecutando en Mininet un ejemplo con VLANs.

```
#Displays all commands available in the app
if usr_input == "commands":

    print("\n\n  enablesw   --> enablesw + {switch id} enables switch as firewal ( 0000000000000001 by default ). ")
    print("  enablesw     --> Enables a switch as a firewall (from rest_firewall) given a switch ID .")
    print("  status       --> Acquires status from all modules.")
    print("  rules        --> Get rules from all switches.")
    print("  config       --> Choose a configuration file to read and create the flows acordingly")
    print("  openex       --> Opens example layout (3 hosts connected to 1 switch )\n")
    print("  openex2      --> Opens example layout with VLAN (4 hosts connected to 1 switch )\n")
    print("  delete       --> Provides a delete curl command template to use.")
    print("  config2rules --> Provides a delete curl command template to use.")
```

Ilustración 19 Listado de código 8

Los tres primeros comandos están relacionados con comandos REST sencillos de Rest_Firewall. En primer lugar, “enablesw” solicita un switch ID (o si pulsas la tecla Enter selecciona el switch ID 0000000000000001 por defecto) y habilita la funcionalidad de “firewall” en ese switch.

```
if usr_input == "enablesw":

    print("\n Write down the switch ID you want to enable (default: 0000000000000001):  \n")
    usr_input2 = input("   >")

    if usr_input2 == "":
        usr_input2 = "0000000000000001"

    crl = pycurl.Curl()
    crl.setopt(crl.URL, 'http://localhost:8080/firewall/module/enable/{}'.format(usr_input2))
    crl.setopt(crl.UPLOAD, 1)
    crl.perform()
    crl.close()
```

Ilustración 20 Listado de código 9

Seguido, “status” es un comando REST que nos muestra qué switches están habilitados como cortafuegos y cuáles no. El tercer comando es “rules”; nos pide un SWID y nos muestra las reglas establecidas en ese switch. En caso de no especificar un SWID nos devuelve las reglas de todos los switches.

```
if usr_input == "status":

    b_obj = BytesIO()
    crl = pycurl.Curl()
    crl.setopt(crl.URL, 'http://localhost:8080/firewall/module/status')
    crl.setopt(crl.WRITEDATA, b_obj)
    crl.perform()
    crl.close()
    get_body = b_obj.getvalue()
    info_str =  get_body.decode('utf8')
    print('Output of GET request:\n\n%s' %info_str)
    print("\n")

if usr_input == "rules":

    print("\n Write down a switch ID or skip to get the rules from all switches:  \n")
    usr_input2 = input("   >")

    if usr_input2 == "":
        usr_input2 = "all"

    b_obj = BytesIO()
    crl = pycurl.Curl()
    crl.setopt(crl.URL, 'http://localhost:8080/firewall/rules/{}'.format(usr_input2))
    crl.setopt(crl.WRITEDATA, b_obj)
    crl.perform()
    crl.close()
    get_body = b_obj.getvalue()
    info_str =  get_body.decode('utf8')
    print('Output of GET request:\n\n%s' %info_str)
    print("\n")
```

Ilustración 21Listado de código 10

Por último “config” y “config2rules” con las dos funcionalidades principales mostradas en los capítulos 4.1.2 y 4.1.3 respectivamente, y “delete” es una función que nos permite eliminar la regla o reglas que deseemos. Al utilizar este comando nos preguntará por la id de la regla que queremos eliminar, el id del switch en el que queremos eliminarla y la id de la vlan en caso de que la red las tenga. Todos estos parámetros pueden contener “all” para eliminar, por ejemplo, todas las reglas de un switch concreto, o de todos los switches. Esta función no pudo ser desarrollada utilizando Pycurl por una limitación que impide añadir un argumento a la hora de usar DELETE, y es por eso que utilizamos la librería Request para poder implementar esta funcionalidad.

4.2 Validación / Evaluación

A continuación, vamos a llevar a cabo dos experimentos distintos para verificar que tanto el formato de la ACL, como la propia aplicación NAC funcionan correctamente. Para ello tomaremos dos ejemplos de la propia documentación de RYU Rest_Firewall [11], pero en vez de configurarlo manualmente a base de comandos REST, crearemos un fichero con el formato de nuestra ACL y lo ejecutaremos en nuestra aplicación. Una vez llevado esto acabo, verificaremos que los flujos se han instalado correctamente y que el comportamiento es el esperado. Por último, utilizaremos la función inversa para generar un archivo de texto con las reglas correspondientes a los flujos actuales, eliminaremos todas las reglas vigentes y cargaremos ese nuevo fichero para verificar que la funcionalidad inversa crea el fichero de forma correcta.

4.2.1 Primera prueba

La primera prueba se hará de forma que configuremos la red que hemos visto previamente en la ilustración 6, pero esta vez desde un archivo con nuestro formato ACL. En primera instancia queremos permitir la comunicación ICMP entre el host 1 y 2, y permitir cualquier comunicación entre el host 2 y 3 exceptuando precisamente el tráfico ICMP. A continuación, mostramos cómo debería de ser el fichero, como también hemos visto anteriormente. Hay que recordar que a pesar de que lo escribamos en mayúsculas, la aplicación NAC admite tanto mayúsculas como minúsculas y una combinación de ambas mientras que el mensaje escrito sea el mismo.

```
NOVLAN
ALL

IPV4  10.0.0.1/32  10.0.0.2/32  -  ICMP  -  -  -     1
IPV4  10.0.0.2/32  10.0.0.3/32  -  ICMP  -  -  DENY  1
IPV4  10.0.0.2/32  10.0.0.3/32  -  -     -  -  -     1
```

Una vez abrimos nuestra aplicación NAC, podemos cargar la topología de este ejemplo en Mininet con el comando “openex” y veremos los 3 flujos que tenemos por defecto en la red (ver la ilustración 22). La primera ordena con la prioridad más alta que se denieguen todos los paquetes, pero esto se debe a que debemos habilitar los switches que queramos como firewalls para “activarlos”, por lo tanto, cuando ejecutamos en nuestra aplicación el comando “enablesw” con el switch id del mismo, se habilita como firewall y ese flujo desaparece como podemos ver la segunda vez que ejecutamos el comando “dump flows” en el switch. Además, como podemos ver en la ilustración 23, también tenemos el feedback de que el switch ha sido habilitado en nuestra aplicación.

```
root@sdnhubvm:~/Desktop/TFG[10:53]$ ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=30.906s, table=0, n_packets=18, n_bytes=1416, priority=65535 actions=drop
 cookie=0x0, duration=30.899s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:128
 cookie=0x0, duration=30.905s, table=0, n_packets=0, n_bytes=0, priority=65534,arp actions=NORMAL
root@sdnhubvm:~/Desktop/TFG[10:53]$ ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=263.948s, table=0, n_packets=0, n_bytes=0, priority=65534,arp actions=NORMAL
 cookie=0x0, duration=263.942s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:128
root@sdnhubvm:~/Desktop/TFG[10:57]$
```

Ilustración 22 Tabla de flujos 1

```
>openex
Terminals opened !

>enablesw

 Write down the switch ID you want to enable (default: 0000000000000001):

   >
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": "
irewall running."}}]>
>
```

Ilustración 23 Captura de la aplicación 1

Los otros dos flujos que tenemos por defecto son el flujo “catch all” que manda al controlador y deniega todos los paquetes que lleguen a este flujo, es por eso por lo que tiene la prioridad 0, y por otro lado tenemos un flujo que permite los paquetes ARP. A continuación, ejecutaremos el comando que leerá el fichero mostrado anteriormente y veremos los flujos que se añaden.

```
>config
Choose a file to read (default: conf.txt)

   >
You are about to import new rules from a file, do you want to delete ALL rules from ever
y switch and vlan before importing new rules ? Type YES to do so. (If you need to delete
 in a more specific way, you can use the delete command)
   >

The configuration has been implemented correctly !


[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "
Rule added. : rule_id=1"}]}][{"switch_id": "0000000000000001", "command_result": [{"resu
lt": "success", "details": "Rule added. : rule_id=2"}]}][{"switch_id": "0000000000000001
", "command_result": [{"result": "success", "details": "Rule added. : rule_id=3"}]}][{"s
witch_id": "0000000000000001", "command_result": [{"result": "success", "details": "Rule
 added. : rule_id=4"}]}][{"switch_id": "0000000000000001", "command_result": [{"result":
 "success", "details": "Rule added. : rule_id=5"}]}][{"switch_id": "0000000000000001", "
command_result": [{"result": "success", "details": "Rule added. : rule_id=6"}]}]>
```

Ilustración 24 Captura de la aplicación 2

```
root@sdnhubvm:~/Desktop/TFG[11:09]$ ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=1000.747s, table=0, n_packets=0, n_bytes=0, priority=65534,arp actions=NORMAL
 cookie=0x1, duration=14.782s, table=0, n_packets=0, n_bytes=0, priority=65530,icmp,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x2, duration=14.782s, table=0, n_packets=0, n_bytes=0, priority=65529,icmp,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=NORMAL
 cookie=0x3, duration=14.780s, table=0, n_packets=0, n_bytes=0, priority=65528,icmp,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=CONTROLLER:128
 cookie=0x4, duration=14.774s, table=0, n_packets=0, n_bytes=0, priority=65527,icmp,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=CONTROLLER:128
 cookie=0x5, duration=14.774s, table=0, n_packets=0, n_bytes=0, priority=65526,ip,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=NORMAL
 cookie=0x6, duration=14.774s, table=0, n_packets=0, n_bytes=0, priority=65525,ip,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x0, duration=1000.741s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:128
root@sdnhubvm:~/Desktop/TFG[11:09]$
```

Ilustración 25 Tabla de flujos 2

Seguidamente, como podemos observas en las figuras 24 y 25, ejecutamos nuestro fichero, en este caso lo tenemos guardado en conf.txt al cual accede por defecto como nos advierte la aplicación, y decidimos no eliminar las reglas anteriores ya que no teníamos ninguna añadida, sólo las que tenemos por defecto. Son precisamente (ver la imagen 25) estas dos normas por defecto las que se quedan en primer y último lugar, ya que se ordenan por prioridad, y las 6 restantes son las que hemos añadido, al tener 3 normas y tener el “1” en la casilla de bidireccional. Por lo tanto, si ejecutamos un “pingall”, es decir, que todos los hosts se hagan un ping entre sí, sólo debería funcionar de h1 a h2 (ver figura 26).

```
mininet> xterm s1
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
h3 -> X X
*** Results: 66% dropped (2/6 received)
```

Ilustración 26 Captura Mininet

El próximo paso que vamos a realizar es utilizar la función inversa para poder guardar la configuración actual en un fichero llamado “RulesToFile.txt” (ver imagen 27). El propio comando nos muestra el contenido final del archivo en la aplicación, y podemos ver que en principio el contenido es correcto. Destacar que a pesar de no ser igual a lo que hemos escrito nosotros en el fichero “conf.txt”, el contenido es equivalente. Nosotros hemos podido utilizar la casilla de bidireccional, y este comando no lo utiliza, y por otro lado nosotros no especificamos IPv4 en la cuarta casilla, ya que es el que se escoge por defecto. Además, nosotros hemos usado ALL y este comando nos devuelve el switch, en esta red sólo hay un switch y es este, por eso en este caso son equivalentes.

```
>config2rules

Rules in file: RulesToFile.txt.

NOVLAN
SWID:0000000000000001
IPV4    10.0.0.1/32    10.0.0.2/32    IPv4    ICMP    -    -    ALLOW    -
IPV4    10.0.0.2/32    10.0.0.1/32    IPv4    ICMP    -    -    ALLOW    -
IPV4    10.0.0.2/32    10.0.0.3/32    IPv4    ICMP    -    -    DENY     -
IPV4    10.0.0.3/32    10.0.0.2/32    IPv4    ICMP    -    -    DENY     -
IPV4    10.0.0.2/32    10.0.0.3/32    IPv4    -    -    -    ALLOW    -
IPV4    10.0.0.3/32    10.0.0.2/32    IPv4    -    -    -    ALLOW    -
```

Ilustración 27 Captura de la aplicación 3

Antes de cargar a nuestro programa las normas desde “RulesToFile.txt”, vamos a hacer una prueba más con las reglas actuales. Vamos a eliminar las dos reglas que bloquean el tráfico ICMP desde el host 2 al 3 y comprobar que una vez las eliminamos, efectivamente se habilitan, por ejemplo, los “pings”. Para ello primero debemos utilizar el comando “rules” en la aplicación para ver las reglas vigentes en el switch y sus respectivas “rule_id” (ver imagen 28). A continuación, eliminaremos con el comando “delete” las normas 3 y 4 que son las mencionadas previamente, y veremos que los flujos son eliminados correctamente.

```
>rules

 Write down a switch ID or skip to get the rules from all switches:

   >
Output of GET request:

[{"access_control_list": [{"rules": [{"priority": 65530, "dl_type": "IPv4", "nw_proto": "ICMP", "nw_dst": "10.
0.0.2", "nw_src": "10.0.0.1", "rule_id": 1, "actions": "ALLOW"}, {"priority": 65529, "dl_type": "IPv4", "nw_pr
oto": "ICMP", "nw_dst": "10.0.0.1", "nw_src": "10.0.0.2", "rule_id": 2, "actions": "ALLOW"}, {"priority": 6552
8, "dl_type": "IPv4", "nw_proto": "ICMP", "nw_dst": "10.0.0.3", "nw_src": "10.0.0.2", "rule_id": 3, "actions":
 "DENY"}, {"priority": 65527, "dl_type": "IPv4", "nw_proto": "ICMP", "nw_dst": "10.0.0.2", "nw_src": "10.0.0.3
", "rule_id": 4, "actions": "DENY"}, {"priority": 65526, "dl_type": "IPv4", "nw_dst": "10.0.0.3", "nw_src": "1
0.0.0.2", "rule_id": 5, "actions": "ALLOW"}, {"priority": 65525, "dl_type": "IPv4", "nw_dst": "10.0.0.2", "nw_
src": "10.0.0.3", "rule_id": 6, "actions": "ALLOW"}]}], "switch_id": "0000000000000001"}]
```

Ilustración 28 Captura de la aplicación 4

Como podemos ver en la imagen 29, eliminamos las reglas correctamente, y en la imagen número 30 verificamos que los flujos han sido eliminados correctamente también. Por último, en la imagen 31 podemos ver cómo los pings llegan desde el host 2 al 3.

```
>delete

Select a rule id to delete (or you can also choose"all" to clear all rules)

   >3
Select a Switch id to delete the rule from (or press Enter to get default: "all")

   >
Select a Vlan id to delete the rule from (or press Enter to get default: none)

   >
Are you sure you want to delete rule id: 3, from switch id: all and vlan id:  ? Type NO if you want to cancel.


   >
Rule(s) deleted!


>delete

Select a rule id to delete (or you can also choose"all" to clear all rules)

   >4
Select a Switch id to delete the rule from (or press Enter to get default: "all")

   >
Select a Vlan id to delete the rule from (or press Enter to get default: none)

   >
Are you sure you want to delete rule id: 4, from switch id: all and vlan id:  ? Type NO if you want to cancel.


   >
Rule(s) deleted!


>rules

 Write down a switch ID or skip to get the rules from all switches:

   >
Output of GET request:

[{"access_control_list": [{"rules": [{"priority": 65530, "dl_type": "IPv4", "nw_proto": "ICMP", "nw_dst": "10.
0.0.2", "nw_src": "10.0.0.1", "rule_id": 1, "actions": "ALLOW"}, {"priority": 65529, "dl_type": "IPv4", "nw_pr
oto": "ICMP", "nw_dst": "10.0.0.1", "nw_src": "10.0.0.2", "rule_id": 2, "actions": "ALLOW"}, {"priority": 6552
6, "dl_type": "IPv4", "nw_dst": "10.0.0.3", "nw_src": "10.0.0.2", "rule_id": 5, "actions": "ALLOW"}, {"priorit
y": 65525, "dl_type": "IPv4", "nw_dst": "10.0.0.2", "nw_src": "10.0.0.3", "rule_id": 6, "actions": "ALLOW"}]}]
, "switch_id": "0000000000000001"}]
```

Ilustración 29 Captura de la aplicación 5

```
root@sdnhubvm:~/Desktop/TFG[11:53]$ ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=2304.285s, table=0, n_packets=12, n_bytes=504, priority=65534,arp actions=NORMAL
 cookie=0x1, duration=2281.960s, table=0, n_packets=2, n_bytes=196, priority=65530,icmp,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x2, duration=2281.960s, table=0, n_packets=2, n_bytes=196, priority=65529,icmp,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=NORMAL
 cookie=0x5, duration=2281.952s, table=0, n_packets=0, n_bytes=0, priority=65526,ip,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=NORMAL
 cookie=0x6, duration=2281.952s, table=0, n_packets=0, n_bytes=0, priority=65525,ip,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x0, duration=2304.284s, table=0, n_packets=2, n_bytes=196, priority=0 actions=CONTROLLER:128
root@sdnhubvm:~/Desktop/TFG[11:53]$
```

Ilustración 30 Tabla de flujos 3

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 h3
h3 -> X h2
*** Results: 33% dropped (4/6 received)
```

Ilustración 31 Captura Mininet 2

Una vez realizadas estas pruebas con éxito, vamos a importar las normas desde el fichero “RulesToFile.txt”, el cual aún contiene las normas de la imagen 27, eliminaremos todas las normas vigentes y veremos si los flujos son los mismos que antes y se han instalado correctamente.

```
>config
Choose a file to read (default: conf.txt)

   >RulesToFile.txt
You are about to import new rules from a file, do you want to delete ALL rules from every switch and vlan befo
re importing new rules ? Type YES to do so. (If you need to delete in a more specific way, you can use the del
ete command)
   >yes
Rule(s) deleted!



The configuration has been implemented correctly !


[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "Rule added. : rule_id=
13"}]}][{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "Rule added. : r
ule_id=14"}]}][{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "Rule add
ed. : rule_id=15"}]}][{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "R
ule added. : rule_id=16"}]}][{"switch_id": "0000000000000001", "command_result": [{"result": "success", "detai
ls": "Rule added. : rule_id=17"}]}][{"switch_id": "0000000000000001", "command_result": [{"result": "success",
 "details": "Rule added. : rule_id=18"}]}]>
>
```

Ilustración 32 Captura de la aplicación 6

```
root@sdnhubvm:~/Desktop/TFG[12:20]$ ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=3999.442s, table=0, n_packets=24, n_bytes=1008, priority=65534,arp actions=NORMAL
 cookie=0xd, duration=3.625s, table=0, n_packets=0, n_bytes=0, priority=65518,icmp,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0xe, duration=3.625s, table=0, n_packets=0, n_bytes=0, priority=65517,icmp,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=NORMAL
 cookie=0xf, duration=3.620s, table=0, n_packets=0, n_bytes=0, priority=65516,icmp,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=CONTROLLER:128
 cookie=0x10, duration=3.615s, table=0, n_packets=0, n_bytes=0, priority=65515,icmp,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=CONTROLLER:128
 cookie=0x11, duration=3.611s, table=0, n_packets=0, n_bytes=0, priority=65514,ip,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=NORMAL
 cookie=0x12, duration=3.611s, table=0, n_packets=0, n_bytes=0, priority=65513,ip,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x0, duration=3999.441s, table=0, n_packets=4, n_bytes=392, priority=0 actions=CONTROLLER:128
root@sdnhubvm:~/Desktop/TFG[12:21]$
```

Ilustración 33 Tabla de flujos 4

4.2.2 Segunda prueba

La segunda prueba que vamos a realizar se trata de la última que encontramos en la documentación de Rest_Firewall [11], para poder testear una red con VLAN y con IPV6. Comenzaremos abriendo nuestra aplicación y utilizando el comando “openex2” el cual nos abrirá directamente la topología de este ejemplo concreto. Podemos ver la red en la ilustración número 34, y el objetivo será habilitar la comunicación ICMPV6 entre el host 1 y 2. Antes de empezar tendremos que utilizar el comando “enablesw” una vez más, y seguir las instrucciones para configurar los hosts de la imagen 35.

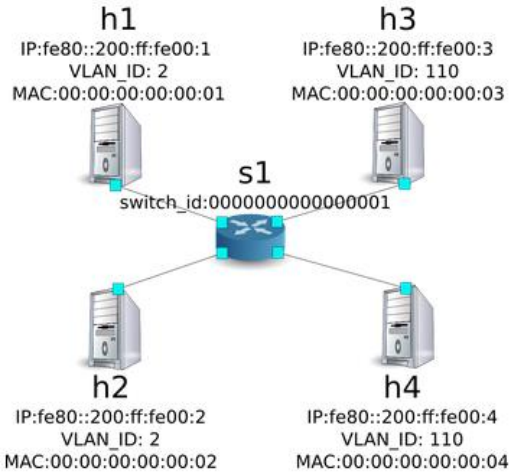

Ilustración 34 Red del ejemplo 2

Next, set the VLAN ID to the interface of each host.

host: h1:

```
# ip addr del fe80::200:ff:fe00:1/64 dev h1-eth0
# ip link add link h1-eth0 name h1-eth0.2 type vlan id 2
# ip addr add fe80::200:ff:fe00:1/64 dev h1-eth0.2
# ip link set dev h1-eth0.2 up
```

host: h2:

```
# ip addr del fe80::200:ff:fe00:2/64 dev h2-eth0
# ip link add link h2-eth0 name h2-eth0.2 type vlan id 2
# ip addr add fe80::200:ff:fe00:2/64 dev h2-eth0.2
# ip link set dev h2-eth0.2 up
```

host: h3:

```
# ip addr del fe80::200:ff:fe00:3/64 dev h3-eth0
# ip link add link h3-eth0 name h3-eth0.110 type vlan id 110
# ip addr add fe80::200:ff:fe00:3/64 dev h3-eth0.110
# ip link set dev h3-eth0.110 up
```

host: h4:

```
# ip addr del fe80::200:ff:fe00:4/64 dev h4-eth0
# ip link add link h4-eth0 name h4-eth0.110 type vlan id 110
# ip addr add fe80::200:ff:fe00:4/64 dev h4-eth0.110
# ip link set dev h4-eth0.110 up
```

Then, set the version of OpenFlow to be used in each router to 1.3.

switch: s1 (root):

```
# ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

Ilustración 35 Configuración del segundo ejemplo

El fichero que vamos a crear, en este caso se llama “conf2.txt”, debe de tener la información que vamos a mostrar a continuación. Destacar que ahora sí debemos especificar la id de la vlan, y como hemos visto en la imagen número 34, los hosts 1 y 2 se encuentran en la vlan 2. Por otro lado, debemos especificar que habilitamos el tráfico ICMPv6 al igual que la IPV6, ya que como hemos visto antes, si lo dejamos vacío escoge IPV4 que es la opción por defecto. Una vez tenemos el fichero preparado, ejecutamos el comando “config” especificando el nombre del fichero (vea la imagen 36) y como podemos comprobar los flujos se han añadido correctamente como en el ejemplo previamente mencionado, y el ping funciona sin problema (vea las imágenes 37 y 38).

VLANID:2
SWID:0000000000000001

IPV6 fe80::200:ff:fe00:1 - IPv6 ICMPv6 - - - -
IPV6 fe80::200:ff:fe00:2 - IPv6 ICMPv6 - - - -

```
>enablesw

 Write down the switch ID you want to enable (default: 0000000000000001):

   >
[{"switch_id": "0000000000000001", "command_result": {"result": "success", "details": "firewall running."}}]>
>
>config
Choose a file to read (default: conf.txt)

   >conf2.txt
You are about to import new rules from a file, do you want to delete ALL rules from every switch and vlan befo
re importing new rules ? Type YES to do so. (If you need to delete in a more specific way, you can use the del
ete command)
   >

The configuration has been implemented correctly !


[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "vlan_id": 2, "details": "Rule add
ed. : rule_id=1"}]}][{"switch_id": "0000000000000001", "command_result": [{"result": "success", "vlan_id": 2,
"details": "Rule added. : rule_id=2"}]}]>
```

Ilustración 36 Captura de la aplicación 7

```
root@sdnhubvm:~/Desktop/TFG[15:49]$ ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=6000.582s, table=0, n_packets=0, n_bytes=0, priority=65534,arp actions=NORMAL
 cookie=0x200000001, duration=5476.435s, table=0, n_packets=6, n_bytes=660, priority=65530,icmp6,dl_vlan=2,ipv6_src=fe80::200:ff:fe00:1 actions=NORMAL
 cookie=0x200000002, duration=5476.435s, table=0, n_packets=6, n_bytes=668, priority=65529,icmp6,dl_vlan=2,ipv6_src=fe80::200:ff:fe00:2 actions=NORMAL
 cookie=0x0, duration=6000.582s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:128
root@sdnhubvm:~/Desktop/TFG[15:50]$
```

Ilustración 37 Tabla de flujos 5

```
root@sdnhubvm:~/Desktop/TFG[14:19]$ ping6 -I h1-eth0.2 fe80::200:ff:fe00:2
PING fe80::200:ff:fe00:2(fe80::200:ff:fe00:2) from fe80::200:ff:fe00:1 h1-eth0.2: 56 data bytes
64 bytes from fe80::200:ff:fe00:2: icmp_seq=1 ttl=64 time=0.267 ms
64 bytes from fe80::200:ff:fe00:2: icmp_seq=2 ttl=64 time=0.040 ms
64 bytes from fe80::200:ff:fe00:2: icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from fe80::200:ff:fe00:2: icmp_seq=4 ttl=64 time=0.063 ms
^C
--- fe80::200:ff:fe00:2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.040/0.103/0.267/0.095 ms
root@sdnhubvm:~/Desktop/TFG[14:20]$
```

Ilustración 38 Captura de Mininet 3

```
>config2rules

Rules in file: RulesToFile.txt.

NOVLAN
SWID:0000000000000001
VLANID:2
SWID:0000000000000001
IPV6    fe80::200:ff:fe00:1    -    IPv6    ICMPv6    -    -    ALLOW    -
IPV6    fe80::200:ff:fe00:2    -    IPv6    ICMPv6    -    -    ALLOW    -
```

Ilustración 39 Captura de la aplicación 8

Por último, si ejecutamos el comando inverso podemos observar en la ilustración 39 que el fichero que generamos es idéntico al que hemos escrito a mano exceptuando las dos primeras filas, las cuales no tienen valor como tal ya que recogería las reglas del switch sin vlan pero en este caso no tenemos ninguna, y es por eso que no aparecen reglas escritas debajo de estas. Consideramos que estos dos ejemplos son adecuados ya que tratamos redes distintas y podemos comprobar que el funcionamiento es el que debería ya que son ejemplos de la propia documentación oficial de RYU.

Capítulo 5. Conclusiones y líneas de trabajo abiertas

Una vez concluido el proyecto podemos confirmar que se han cumplido los objetivos de este, ya que se ha creado con éxito una aplicación NAC, adecuada a entornos IoT, sin utilizar la logística y los recursos que son requeridos para un sistema de credenciales. De esta forma obtenemos una herramienta sencilla de utilizar, de poner en marcha y bastante potente. Además, se ha logrado un conocimiento mayor sobre diversas ramas en relación con la seguridad de redes y sistemas SDN OpenFlow.

También se han adquirido conocimientos básicos sobre ACL entre otros, y se han puesto en práctica diseñando una, adecuada a la API de RYU Rest_Firewall, además de conseguir más soltura utilizando herramientas como Mininet.

Una de las conclusiones a las que llegué, fue que elegir Python fue un acierto. A pesar de no tener a penas experiencia con este, mi tutor me lo recomendó en vez de utilizar C o C++, y tras haberle dedicado tiempo, lo cierto es que la forma de programar en este lenguaje me parece muy intuitiva y completa. Me alegro de haber tomado esa decisión y ganar experiencia con este lenguaje.

Por otro lado, cabe subrayar algunas líneas de trabajo abiertas que podrían ser pulidas o añadidas en caso de contar con más tiempo, como, por ejemplo, añadir Rest_Topology (una aplicación RYU la cual nos proporciona información de la topología de la red y de los elementos de la misma) al proyecto para poder tener otra forma potente de interrogar los switches y poder obtener información valiosa de cara a monitorizar la red.

Por otro lado, el poder refinar las reglas de Rest_Firewall podría ser otro punto importante, como por ejemplo poder especificar los tipos de ICMP que se quieren permitir o bloquear. Además de eso una idea importante pero que se tuvo que descartar por falta de tiempo fue el poder controlar también el ancho de banda de los hosts.

Capítulo 6. Planificación y gestión

6.1 Planificación del proyecto

6.1.1 Estructura de descomposición de trabajo

A continuación, se expondrá la planificación seguida para realizar el TFG, y para ello utilizaremos una estructura de descomposición de trabajo. Se trata de una herramienta cuya finalidad consiste en la descomposición jerárquica del proyecto y nos servirá también para definir el alcance de este. Los paquetes de trabajo son los elementos situados al final de dicha jerarquía, los cuales cuentan con una o dos letras a modo de abreviación. Podemos observarlo en la figura 40:

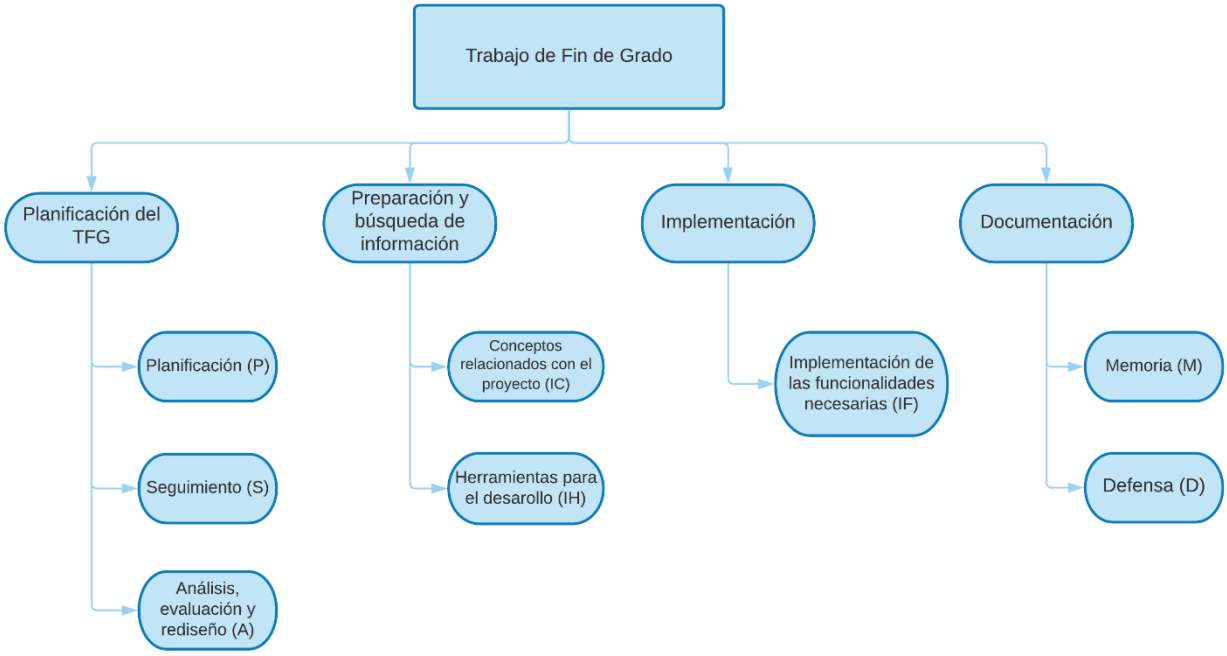

Ilustración 40 Estructura de descomposición de trabajo del TFG

6.1.2 Paquetes de trabajo

Un paquete de trabajo es una descripción de una operación o tarea que va a llevarse a cabo dentro del proyecto. A continuación, vamos a desglosar los paquetes de trabajo presentados.

- Paquete de planificación del TFG:

- Planificación (P): Este paquete engloba la planificación previa a la realización del proyecto, la creación de los propios paquetes de trabajo y su duración estimada.

- Seguimiento (S): Se realizará un seguimiento del TFG, para corroborar el avance de este y a su vez identificar desviaciones de la planificación original.

- Análisis, evaluación y rediseño (A): El objetivo de este paquete es analizar que la planificación inicial es adecuada, y poder realizar algún cambio en la misma en el caso de encontrar una desviación que no permita llevar a cabo el planteamiento estudiado en un principio.

- Paquete de preparación y búsqueda de información:

  - Conceptos relacionados con el proyecto (IC):

    - IC1. NAC: Se buscará información a cerca de los NAC ("Network Access Control") para aprender las basas y diseñar uno el cual cumpla los requisitos de nuestro proyecto.

    - IC2. ACL: Obtención y comprensión de información relacionada con ACL ("Access Control List").

    - IC3. SDN: Recopilaremos información acerca de los SDN ("Software-Defined Networking").

    - IC4. OpenFlow: Se adquirirán los conocimientos necesarios a cerca de OpenFlow y de su estructura.

  - Herramientas para el desarrollo (IH):

    - IH1. Mininet: Buscaremos información a cerca de la estructura, la utilización y la manera de instalar Mininet para poder utilizarlo a modo de entorno de experimentación del proyecto.

    - IH2. RYU: Nos informaremos a cerca del framework de SDN-OpenFlow llamado RYU, el cual utilizaremos en el TFG.

    - IH3. Python: Se investigarán librerías de Python que faciliten el desarrollo de nuestra aplicación NAC y habiliten las funcionalidades planificadas.

- Paquete de implementación:

- Implementación de las funcionalidades necesarias (IF):
    - IF1. Estructura ACL: Se definirá una estructura ACL adecuada para nuestra aplicación NAC.
    - IF2. Configuración desde fichero: Se desarrollará una funcionalidad la cual nos permita cargar reglas a nuestra red definidas en un fichero con la estructura ACL mencionada previamente.
    - IF3. Proceso Inverso: Desarrollaremos una funcionalidad la cual obtenga los flujos vigentes actualmente y cree un fichero de texto con el listado de reglas que las componen. Este fichero seguirá la estructura ACL.
    - IF4. Funcionalidades secundarias: Se implementarán funcionalidades que nos permitan realizar acciones comunes como eliminar reglas, conseguir información a cerca de las reglas vigentes, habilitar o deshabilitar switches desde la propia consola de nuestra aplicación NAC.

- Paquete de Documentación:
    - Memoria (M): Se realizará la memoria final del TFG.
    - Defensa (D): Preparación de la defensa del TFG.

6.1.3 Tiempo estimado de desarrollo

A continuación, se muestra una tabla con el tiempo dedicado a cada paquete de forma estimada (ver la imagen 41).

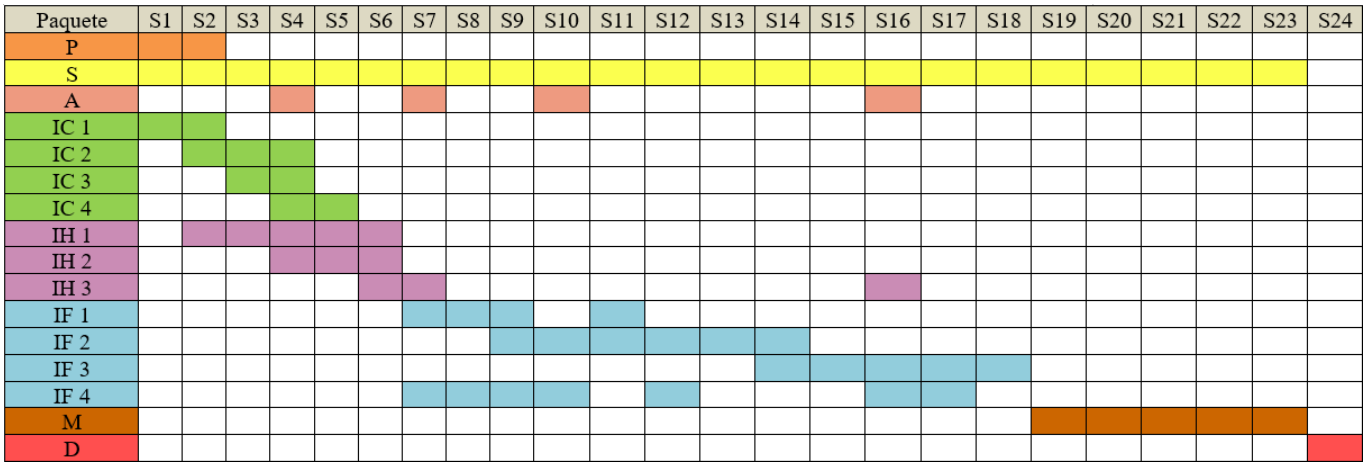

Ilustración 41Tabla de tiempo dedicado

6.2 Desviaciones

Una de las desviaciones más importantes fue realizada al comienzo, ya que la atención del desarrollo del proyecto fue dirigida a partir de la base de Rest_Router (una aplicación RYU la cual se centra en darle a los switches un comportamiento parecido a un router), añadiendo varias API-s RYU y resultó no ser la forma correcta de empezar, ya que posteriormente se identificó que partir desde Rest_Firewall iba a ser lo más sensato.

Por otro lado, otra desviación menor fue la limitación de PyCurl, el cual como ya se ha mencionado previamente, no era posible añadir un argumento al comando DELETE desde la aplicación NAC. Aún así este problema fue solventado posteriormente gracias a la librería Requests que sí que nos permite añadir argumentos a una orden de DELETE, y es así como se ha conseguido completar el comando “delete” de nuestra aplicación el cual nos permite eliminar la o las reglas que deseemos.

Capítulo 7. Bibliografía

[1] Hawei, What Is an Access Control List (ACL) https://support.huawei.com/enterprise/en/doc/EDOC1100086647 Visitado el 07/2021

[2] Wikipedia, SDN https://en.wikipedia.org/wiki/Software-defined_networking Visitado 07/2021

[3] OpenNAC, OpenCloudFactory https://www.opencloudfactory.com/opennac-enterprise/ Visitado el 07/2021

[4] Network access control and management solution, Xavier González, Octubre 2012. https://wiki.centos.org/Events/Dojo/Madrid2013?action=AttachFile&do=get&target=overview_opennac_org_eng_v9.pdf

[5] Wikipedia, Network Access Control https://en.wikipedia.org/wiki/Network_Access_Control Visto el 07/2021

[6] Wikipedia, Software Defined Network https://es.wikipedia.org/wiki/Redes_definidas_por_software Visitado el 07/2021

[7] Página principal de Mininet: http://mininet.org/overview/ Visitado el 09/2020

[8] Página de Descarga Mininet: http://mininet.org/download/ Visitado el 09/2020

[9] Wikipedia Virtual Environment Software: https://en.wikipedia.org/wiki/Virtual_environment_software Visitado el 07/2021

[10] RYU Documentación: https://osrg.github.io/ryu-book/en/html/index.html Visitado el09/2020

[11] RYU Doc. Rest_Firewall https://osrg.github.io/ryu-book/en/html/rest_firewall.html#example-of-operation-of-a-single-tenant-ipv4 Visitado el 10/2020

[12] RYU Doc. Ofctl_Rest https://ryu.readthedocs.io/en/latest/app/ofctl_rest.html Visitado el 07/2021

[13] Wikipedia Open vSwitch https://es.wikipedia.org/wiki/Open_vSwitch Visitado el 07/2021

[14] Wikipedia OpenFlow https://es.wikipedia.org/wiki/OpenFlow Visitado el 07/2021

[15] Wikipedia REST https://en.wikipedia.org/wiki/Representational_state_transfer Visto 07/2021

[16] Curl, command line tool and library for transferring data and URLs (since 1998) https://curl.se/ Visitado el 07/2021

[17] Requests: HTTP para Humanos, Kenneth Reitz (2013) https://docs.python-requests.org/es/latest/ Visitado el 07/2021

[18] Página oficial de RYU https://ryu-sdn.org/ Visto el 09/2020

[I_1] Imagen de red local tomada de https://www.lucidchart.com/pages/es/como-dibujar-un-diagrama-de-red Visitado el 07/2021

[I_2] Imagen original: https://support.huawei.com/enterprise/en/doc/EDOC1100086647 Visitado el 07/2021

[I_3] Imagen original: https://support.huawei.com/enterprise/en/doc/EDOC1100086647 Visitado el 07/2021

[I_4] Imagen original tomada de: Younus, Muhammad & Islam, Saif & Kim, Sung Won. (2019). Proposition and Real-Time Implementation of an Energy-Aware Routing Protocol for a Software Defined Wireless Sensor Network. Sensors. 19. 2739. 10.3390/s19122739. https://www.researchgate.net/figure/Software-defined-networking-SDN-architecture_fig1_333873385 Visitado el 07/2021

[I_5] Imagen original: https://opennetworking.org/mininet/ Visitado el 07/2021

[I_6] Imagen original: https://osrg.github.io/ryu-book/en/html/rest_firewall.html#example-of-operation-of-a-single-tenant-ipv4 Visitado el 02/2021

Capítulo 8. Anexos

A continuación, vamos a dar las pautas a seguir para poder replicar el entorno de experimentación utilizado, su puesta en marcha, comandos que pueden ser de gran utilidad. De esta forma se podrá poner en marcha tanto la aplicación NAC, como los ejemplos expuestos previamente. Además, adjuntaremos el código de la aplicación desarrollada, para que sea más accesible.

8.1 Preparación del entorno de trabajo

El primer paso será poner en marcha el entorno de trabajo con Mininet, y la propia página oficial recomienda descargar una máquina virtual con Ubuntu y Mininet como podemos ver en el apartado de descargas de la misma [7]. Seguiremos los pasos recomendados para poder poner en marcha la máquina virtual, y ejecutaremos los siguientes comandos para terminar de configurar y personalizar el entorno, como por ejemplo cambiar el idioma del teclado al castellano y poder utilizar startx y contar con un entorno gráfico.

```
sudo loadkeys es – pone el teclado en castellano, en modo texto
sudo dhclient eth0 (host only, 192.168…)
sudo dhclient eth1 (NAT, 10.0…)
sudo apt-get update
sudo apt-get install xinit lxde virtualbox-guest-dkms
```

Tras estos comandos, podremos utilizar startx para lanzar el gestor de ventanas lxde. El siguiente paso será instalar RYU, y para ello utilizaremos los comandos que vamos a ver a continuación, pero es importante recordar que en la página principal de RYU [18] también se explica cómo instalarlo. En nuestro caso particular, la clave para instalar correctamente RYU fue instalar pip para python3:

```
sudo apt-get update
sudo apt-get install python3-pip
sudo apt-get install gcc libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib1g-dev

sudo pip3 install ryu
```

8.2 Comandos más frecuentes

Una vez instalado correctamente el entorno de experimentación, podemos empezar a hacer pruebas tanto con Mininet, como con las distintas APIs que nos proporciona RYU. Además, podemos poner en marcha los ejemplos mencionados en esta memoria junto a la aplicación NAC desarrollada. Ahora mencionaremos algunos comandos fuera de nuestra aplicación que facilitará el uso de este entorno de trabajo.

- Sudo mn -c | Este comando limpia todos los procesos de Mininet.

- sudo mn --topo single,3 --mac --switch ovsk --controller remote | Este comando sirve para abrir en Mininet la topología de la figura número 6. En este caso "--topo" sirve para generar una topología, y "single,3" nos indica que tendremos un switch conectado a 3 hosts. "--mac" hará que las direcciones MAC sean seguidas en vez de aleatorias empezando desde "00:00:...:01", y "controller" nos indica que queremos que el controlador sea remoto (para poder utilizar, por ejemplo, Rest_Firewall).

- xterm s1 | Una vez hayamos ejecutado una topología, podemos utilizar el comando xterm para abriri una ventana con el dispositivo que queramos, en este ejemplo accederíamos al switch 1, pero también podemos utilizarlo con hosts para hacer ping a otros.

- ryu-manager ryu.app.rest_firewall | Este comando se utiliza para ejecutar una aplicación RYU, en este cas rest_firewall, pero podemos escoger cualquier aplicación que nos trae RYU por defecto o incluso aplicaciones que desarrollemos nosotros mismos. Para ello debemos utilizar el comando "xterm c0" desde Mininet, para acceder al controlado, y ejecutar ahí este comando.

- ovs-ofctl -O openflow13 dump-flows s1| Podemos utilizar este comando en un switch tras haber accedido a él vía el comando "xterm" para conseguir la tabla de flujos de dicho switch.

- Pingall | Este comando hace un testeo de pings desde cada host al resto de hosts de la red.

- nodes, dump | Nodes es un comando de Mininet que nos devuelve la lista de nodos (hosts, switches, controladores) de la topología, y por otro lado dump es un comando parecido pero que nos proporciona información más detallada.

Estos son los comandos más frecuentados, por la información que proporcionan y también por su uso de cara a experimentar con distintas topologías y controladores. En general, tras haber logrado experiencia con Mininet y RYU sentimos que son herramientas idóneas de cara a testear, simular y crear aplicaciones orientadas a redes reales.

8.3 Código de la aplicación NAC

Por último, se adjuntará el código de nuestra aplicación NAC para facilitar el acceso al mismo. Para los comandos "openex" y "openex2" veremos que se menciona un fichero "custom_rest_firewall.py" que es simplemente un fichero donde se juntan Rest_Firewall y Ofctl_rest. Este fichero es útil para agilizar la puesta en marcha de los ejemplos, pero no es necesario, ya que podemos abrir estas aplicaciones RYU sin problema como ya hemos visto previamente.

```
from urllib.parse import urlencode
import pycurl, json, os, re, requests, socket
from io import BytesIO

#Cheks if the ipv4 address is correct (with mask)
def check_ipv4(ip):
  #mask
  str = ip.split("/")
  if len(str) != 2:
    return False

  i = int(str[1])
  if i < 0 or i > 32:
    return False
  #ip
  try:
    socket.inet_aton(str[0])
    return True
  except socket.error:
    return False

#Cheks if the ipv6 address is correct
def check_ipv6(ip):
  try:
    socket.inet_pton(socket.AF_INET6, ip)
    return True
  except socket.error:
    return False

def check_mac(mac):

  if re.match("[0-9a-f]{2}([-:]?)[0-9a-f]{2}(\\1[0-9a-
f]{2}){4}$", mac.lower()):
    return True
  else:
    return False

def get_next(str, l):

  if str in l:
    aux = l.split(str)
    aux = aux[1].split("")
    aux = aux[2]
    return aux
  else:
    return " "

def get_next2(str, l):

  if str in l:
    aux = l.split(str)
    aux = aux[1].split(',')
    aux = aux[0]
    return aux[3:]
  else:
    return " "

priority = 65530
usr_input = ""

#Main Program

print("Wellcome ! You can type -> commands to get
the command list\n")

while usr_input != "exit":
  usr_input = input(">")

  #Displays all commands available in the app
  if usr_input == "commands":

    print("\n\n  enablesw   --> enablesw + {switch id}
enables switch as firewal ( 0000000000000001 by
default ). ")
    print("  status       --> Acquires status from all
modules.")
    print("  rules        --> Get rules from all
switches.")
    print("  config       --> Choose a configuration file
to read and create the flows acordingly")
    print("  openex       --> Opens example layout (3
hosts connected to 1 switch )\n")
    print("  openex2      --> Opens example layout
with VLAN (4 hosts connected to 1 switch )\n")
    print("  delete       --> Provides a delete curl
command template to use.")
    print("  config2rules --> Provides a delete curl
command template to use.")


  if usr_input == "openex":

    c1 = "xterm -e 'bash -c \"cd /home/ubuntu/ryu
&& ./bin/ryu-manager --verbose
/home/ubuntu/Desktop/TFG/custom_rest_firewall.py
; exec bash\"'"
    c2 = "xterm -e 'bash -c \"sudo mn --topo single,3 -
-mac --switch ovsk --controller remote ; exec bash\"'"
    os.popen(c1,'r',1)
    os.popen(c2,'r',1)
    print("Terminals opened !\n")

  if usr_input == "openex2":

    c1 = "xterm -e 'bash -c \"cd /home/ubuntu/ryu
&& ./bin/ryu-manager --verbose
/home/ubuntu/Desktop/TFG/custom_rest_firewall.py
; exec bash\"'"
    c2 = "xterm -e 'bash -c \"sudo mn --topo single,4 -
-mac --switch ovsk --controller remote -x ; exec
bash\"'"
    os.popen(c1,'r',1)
    os.popen(c2,'r',1)
    print("Terminals opened !\n")

  if usr_input == "enablesw":

    print("\n Write down the switch ID you want to
enable (default: 0000000000000001):  \n")
    usr_input2 = input("   >")

    if usr_input2 == "":
      usr_input2 = "0000000000000001"

    crl = pycurl.Curl()
    crl.setopt(crl.URL,
'http://localhost:8080/firewall/module/enable/{}'.form
at(usr_input2))
    crl.setopt(crl.UPLOAD, 1)
```

```
    crl.perform()
    crl.close()


  if usr_input == "status":

    b_obj = BytesIO()
    crl = pycurl.Curl()
    crl.setopt(crl.URL,
'http://localhost:8080/firewall/module/status')
    crl.setopt(crl.WRITEDATA, b_obj)
    crl.perform()
    crl.close()
    get_body = b_obj.getvalue()
    info_str =  get_body.decode('utf8')
    print('Output of GET request:\n\n%s' %info_str)
    print("\n")

  if usr_input == "rules":

    print("\n Write down a switch ID or skip to get
the rules from all switches:  \n")
    usr_input2 = input("   >")

    if usr_input2 == "":
      usr_input2 = "all"

    b_obj = BytesIO()
    crl = pycurl.Curl()
    crl.setopt(crl.URL,
'http://localhost:8080/firewall/rules/{}'.format(usr_in
put2))
    crl.setopt(crl.WRITEDATA, b_obj)
    crl.perform()
    crl.close()
    get_body = b_obj.getvalue()
    info_str =  get_body.decode('utf8')
    print('Output of GET request:\n\n%s' %info_str)
    print("\n")

  if usr_input == "delete":

    print("\nSelect a rule id to delete (or you can also
choose""all"" to clear all rules)\n")
    id = input("   >")

    print("Select a Switch id to delete the rule from
(or press Enter to get default: """all""")\n")
    usr_input2 = input("   >")
    if usr_input2 == "":
      swid = "all"
    else:
      swid = usr_input2

    print("Select a Vlan id to delete the rule from (or
press Enter to get default: none)\n")
    usr_input2 = input("   >")
    if usr_input2 == "":
      vlanid = ""
    else:
      vlanid = ("/" + usr_input2)

    print("Are you sure you want to delete rule id: {},
from switch id: {} and vlan id: {} ? Type NO if you
want to cancel. \n".format(id, swid, vlanid))
    usr_input2 = input("   >")

    if usr_input2.upper() != "NO":

      mydata = {}
      mydata['rule_id'] = id
      URL_delete =
"http://localhost:8080/firewall/rules/{}{}".format(swi
d,vlanid)
      r = requests.delete(URL_delete,
data=json.dumps(mydata))
      print("Rule(s) deleted!\n\n")
    else:
      print("You have exited the DELETE
command")

  if usr_input == "config2rules":

    #curl -X GET http://localhost:8080/stats/switches
    b_obj = BytesIO()
    crl = pycurl.Curl()
    crl.setopt(crl.URL,
'http://localhost:8080/stats/switches')
    crl.setopt(crl.WRITEDATA, b_obj)
    crl.perform()
    get_body = b_obj.getvalue()
    info_str =  get_body.decode('utf8')

    info_str = info_str[1:-1]
    info_str = info_str.replace(" ","")
    sw_list = info_str.split(",")

    f = open("RulesToFile.txt","w")

    for i in sw_list:
      #curl -X GET
http://localhost:8080/stats/flow/{dpid}
      crl.setopt(crl.URL,
'http://localhost:8080/stats/flow/{}'.format(i))
      crl.setopt(crl.WRITEDATA, b_obj)
      crl.perform()
      get_body = b_obj.getvalue()
      info_str =  get_body.decode('utf8')

      atr = ["-"]*8
      vlan = " "#VLANID:xx or NOVLAN
      swaux = "0000000000000001"
      swid = "SWID:" + swaux[:-len(i)] + i

      for l in info_str.split("actions"):
        #print("{}\n".format(l))#delete first "#" to
print each segment
        #If vlan is different we print vlan and switchid
        aux = " "
        if "vlan" in l:
          aux = l.split("dl_vlan")
          aux = aux[1].split("")
          aux = "VLANID:" + aux[2]

        else:
          aux = "NOVLAN"

        if aux != vlan:
          f.write("{}\n{}\n".format(aux,swid))
          vlan = aux
```

```
#rule in fragment:
#src
atr = ["-"]*8
if "dl_src" in l:
  atr[0] = "MAC"
  atr[1] = get_next("dl_src",l)

elif "nw_src" in l:
  atr[0] = "IPV4"
  src = get_next("nw_src",l)
  if "/" in src:
    atr[1] = src
  else:
    atr[1] = src + "/32"

elif "ipv6_src" in l:
  atr[0]= "IPV6"
  atr[1] = get_next("ipv6_src",l)

elif "in_port" in l:
  atr[0]= "PORT"
  atr[1] = get_next("in_port",l)
#dst
if "dl_dst" in l:
  atr[2] = get_next("dl_dst",l)

elif "nw_dst" in l:
  dst = get_next("nw_dst",l)
  if "/" in src:
    atr[2] = dst
  else:
    atr[2] = dst + "/32"

elif "ipv6_dst" in l:
  atr[2] = get_next("ipv6_dst",l)

#dl_type
if "dl_type" in l:
  aux = get_next2("dl_type",l)
  if aux == "2048":
   atr[3] = "IPv4"
  elif aux == "34525":
   atr[3] = "IPv6"
  else: #2054
    atr[3] = "ARP"

#nw_proto
if "nw_proto" in l:
  aux = get_next2("nw_proto",l)
  if aux == "1":
   atr[4] = "ICMP"
  elif aux == "58":
   atr[4] = "ICMPv6"
  elif aux == "17":
   atr[4] = "UDP"
  else:
   atr[4] = "TCP"

#tp_src
if "tp_src" in l:
  atr[5] = get_next("tp_src",l)

#tp_dst
if "tp_dst" in l:
  atr[6] = get_next("tp_dst",l)

        #Actions : Allow/normal ->
OUTPUT:4294967290 || Deny/controller ->
OUTPUT:4$
        if "OUTPUT:4294967293" in l:
          atr[7] = "DENY"
        else:
          atr[7] = "ALLOW"

        if atr[0] != "-":
          f.write("{}  {}  {}  {}  {}  {}  {}  {}  -
\n".format(atr[0], atr[1], atr[2], atr[3], atr[4], atr[5],
atr[6], atr[7],))

    ###
    f.close()
    f = open("RulesToFile.txt","r")
    print("\nRules in file:
RulesToFile.txt.\n\n{}\n".format(f.read()))
    f.close()

    crl.close()


  if usr_input == "config":

    print("Choose a file to read (default: conf.txt)
\n")
    usr_input2 = input("   >")

    if usr_input2 == "":
      defaultfile = "conf.txt"
    else:
      defaultfile = usr_input2

    #Test phase
    try:
      file = open("{}".format(defaultfile), "r")
    except:
      print("\n File not found ERROR ! \n")
      break

    types = ["dl_src", "dl_dst", "in_port", "nw_src",
"nw_dst", "ipv6_src", "ipv6_dst"]
    attributes = [" "]*9

    swid = "all"
    vlan = " "
    count = 0 # amount of attributes
    linec = 1 # counts the ammount of lines
    flag = 0

    for line in file:

      for word in line.split():
        attributes[count] = word
        count += 1

      if count == 1: #change switch ID or VLAN
        if word.upper() == "NOVLAN":
          vlan = " "

        elif "VLANID:" in word.upper():
          id = word.split(":")
          vlan = ("/" + id[1])
```

```
        elif word.upper() == "ALL":
          swid = "all"

        elif "SWID:" in word.upper():
          id = word.split(":")
          swid = id[1]

        else:
          print("Syntax ERROR at line
{}".format(linec))
          break

      if count > 1:
        #Typpe
        t = 0
        if attributes[0].upper() == "MAC":
          t = 0
        elif attributes[0].upper() == "PORT":
          t = 2
        elif attributes[0].upper() == "IPV4":
          t = 3
        elif attributes[0].upper() == "IPV6":
          t = 5

        #First and second attribute SRC
        if t == 0:
          if not(check_mac(attributes[1])):
            print("Syntax error, MAC structure
expected in Source field at line: {}\n".format(linec))
            flag = 1
          if attributes[2] != "-": #!! Double if because
F and F = T but we only want it to enter if both are T
            if not(check_mac(attributes[2])):
              print("Syntax error, MAC structure
expected in Destination field at line:
{}\n".format(linec))
              flag = 1

        if t == 3:
          if not(check_ipv4(attributes[1])):
            print("Syntax error, IPV4 structure
expected (with mask) in Source field at line:
{}\n".format(linec))
            flag = 1
          if attributes[2] != "-":
            if not(check_ipv4(attributes[2])):
              print("Syntax error, IPV4 structure
expected (with mask) in Destination field at line:
{}\n".format(linec))
              flag = 1

        if t == 5:
          if not(check_ipv6(attributes[1])):
            print("Syntax error, IPV6 structure
expected in Source field at line: {}\n".format(linec))
            flag = 1
          if attributes[2] != "-":
            if not(check_ipv6(attributes[2])):
              print("Syntax error, IPV6 structure
expected in Destination field at line:
{}\n".format(linec))
              flag = 1

        if t == 2:
          try:
            i = int(attributes[1])
          except:
            print("Syntax error at line: {}, in_port
must be in range [0-65535] and must be a number
\n".format(linec))
            flag = 1
            break
          if i < 0 or 1 > 65535 :
            print("Syntax error at line: {}, in_port
must be in range [0-65535] \n".format(linec))
            flag = 1

          if attributes[2] != "-":
            if not(check_mac(attributes[2]) or
check_ipv4(attributes[2]) or
check_ipv6(attributes[2])):
              print("Syntax error, MAC, IPV4 or
IPV6 structure expected in Destination field at line:
{}\n".format(linec))
              flag = 1


        #Third Attribute (dl type)
        if attributes[3] != "-":
          if not(attributes[3].upper() in {"ARP",
"IPV4", "IPV6"}):
            print("Syntax error at line {}, dl_type field
must be: ARP, IPv4 or IPv6\n".format(linec))
            flag = 1

        #Forth Attribute (nw proto)
        if attributes[4] != "-":
          if not(attributes[4].upper() in {"TCP",
"UDP", "ICMP", "ICMPV6"}):
            print("Syntax error at line {}, nw_proto
field must be: TCP, UDP, ICMP or
ICMPv6\n".format(linec))
            flag = 1
          #tp_src and dst can only appear with UDP
or TCP, not with ICMP
          if attributes[4].upper() in {"ICMP",
"ICMPV6"}:
            if attributes[5] != "-" or attributes[6] != "-
":
              print("Error at line {}, the protocol must
be TCP or UDP in order to have a transport protocol
src and dst.\n".format(linec))
              flag = 1


        #Fifth Attribute (tp_src)
        if attributes[5] != "-":
          try:
            i = int(attributes[5])
          except:
            print("Syntax error at line: {}, tp_src must
be in range [0-65535] and must be a number
\n".format(linec))
            flag = 1
            break
          if i < 0 or i > 65535 :
            print("Syntax error at line: {}, tp_src must
be in range [0-65535] \n".format(linec))
            flag = 1
```

```
        #Sixth Attribute (tp_dst)
        if attributes[6] != "-":
          try:
            i = int(attributes[6])
          except:
            print("Syntax error at line: {}, tp_dst must
be in range [0-65535] and must be a number
\n".format(linec))
            flag = 1
            break
          if i < 0 or i > 65535:
            print("Syntax error at line: {}, tp_dst dst
be in range [0-65535] \n".format(linec))
            flag = 1

        #Seventh Attribute (actions)
        if attributes[7] != "-":
          if not(attributes[7].upper() in {"ALLOW",
"DENY"}) :
            print("Syntax error at line {}, actions field
must be: ALLOW or DENY\n".format(linec))
            flag = 1

      count = 0
      linec += 1

    file.close()
    if flag == 1:
      break

    #Execute phase

    #First we will ask if the user wants to delete all
previous rules
    print("You are about to import new rules from a
file, do you want to delete ALL rules from every
switch and vlan before importing new rules ? Type
YES to do so. (If you need to delete in a more
specific way, you can use the delete command)")
    usr_input2 = input("   >")

    if usr_input2.upper() == "YES":

      mydata = {}
      mydata['rule_id'] = "all"
      URL_delete =
"http://localhost:8080/firewall/rules/all/all"
      r = requests.delete(URL_delete,
data=json.dumps(mydata))
      print("Rule(s) deleted!\n\n")

    try:
      file = open("{}".format(defaultfile), "r")
    except:
      print("\n File not found ERROR ! \n")
      break

    crl = pycurl.Curl()

    swid = "all"
    vlan = " "
    count = 0 # amount of attributes
    linec = 1 # counts the ammount of lines
    for line in file:

      for word in line.split():
        attributes[count] = word
        count += 1

      if count == 1: #change switch ID or VLAN
        if word.upper() == "NOVLAN":
          vlan = " "

        elif "VLANID:" in word.upper():
          id = word.split(":")
          vlan = ("/" + id[1])

        elif word.upper() == "ALL":
          swid = "all"

        elif "SWID:" in word.upper():
          id = word.split(":")
          swid = id[1]

        else:
          print("Syntax ERROR at line
{}".format(linec))
          break

      crl.setopt(crl.URL,
'http://localhost:8080/firewall/rules/{}{}'.format(swid
,vlan))

      if count > 1:

        #Typpe
        t = 0
        if attributes[0].upper() == "MAC":
          t = 0
        elif attributes[0].upper() == "PORT":
          t = 2
        elif attributes[0].upper() == "IPV4":
          t = 3
        elif attributes[0].upper() == "IPV6":
          t = 5

        #First Attribute SRC
        data = {"{}".format(types[t]) :
"{}".format(attributes[1])}

        #Second Attribute DST
        if attributes[2] != "-":

          if t == 2: #PORT dst can be
MAC/IPV4/IPV6
            if check_mac(attributes[2]):
              data["{}".format(types[1])] =
"{}".format(attributes[2])

            elif check_ipv4(attributes[2]):
              data["{}".format(types[4])] =
"{}".format(attributes[2])

            elif check_ipv6(attributes[2]):
              data["{}".format(types[6])] =
"{}".format(attributes[2])

          else:
```

```
            data["{}".format(types[t+1])] =
"{}".format(attributes[2])

        #Third Attribute (dl type)
        if attributes[3] != "-":
          if attributes[3].upper() == "ARP":
            data["dl_type"] = "{}".format("ARP")
          elif attributes[3].upper() == "IPV4":
            data["dl_type"] = "{}".format("IPv4")
          elif attributes[3].upper() == "IPV6":
            data["dl_type"] = "{}".format("IPv6")

        #Forth Attribute (nw proto)
        if attributes[4] != "-":
          if attributes[4].upper() == "ICMPV6":
            data["nw_proto"] =
"{}".format("ICMPv6")
          else:
            data["nw_proto"] =
"{}".format(attributes[4].upper())

        #Fifth Attribute (tp_src)
        if attributes[5] != "-":
          data["tp_src"] = "{}".format(attributes[5])

        #Sixth Attribute (tp_dst)
        if attributes[6] != "-":
          data["tp_dst"] = "{}".format(attributes[6])

        #Seventh Attribute (actions)
        if attributes[7] != "-":
          data["actions"] =
"{}".format(attributes[7].upper())

        #Priority Automatically added
        data["priority"] = "{}".format(priority)
        priority -= 1


        #print("DATA: {}\n\n".format(data))
        pf = json.dumps(data)
        crl.setopt(crl.POSTFIELDS, pf)
        crl.perform()

        if attributes[8] == "1": #Bidirectional
          data["{}".format(types[t])] =
"{}".format(attributes[2])
          data["{}".format(types[t+1])] =
"{}".format(attributes[1])
          data["priority"] = "{}".format(priority)
          priority -= 1
          #print("DATA: {}".format(data))
          pf = json.dumps(data)
          crl.setopt(crl.POSTFIELDS, pf)
          crl.perform()

      count = 0
      linec += 1

    file.close()
    crl.close()
    print("\nThe configuration has been implemented
correctly ! \n\n")
#Whl/
print ("Exiting App !")
```