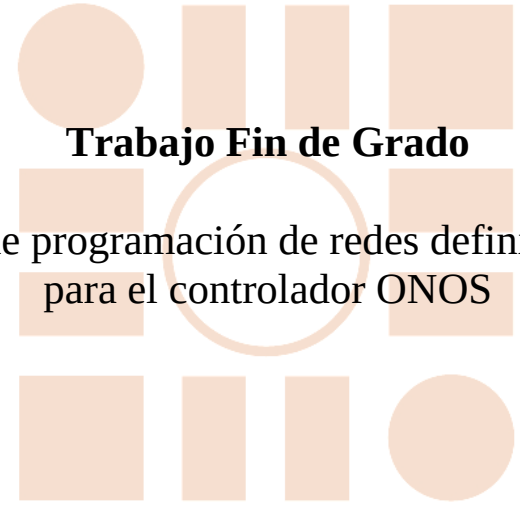
Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA



Trabajo Fin de Grado

Interfaz visual de programación de redes definidas por software para el controlador ONOS

Autor: Álvaro Sánchez López

Tutor/es: Elisa Rojas Sánchez

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo Fin de Grado

Interfaz visual de programación de redes definidas por software para el controlador ONOS

Autor: Álvaro Sánchez López

Tutor/es: Elisa Rojas Sánchez

TRIBUNAL:

Presidente: Isaías Martínez Yelmo

Vocal 1º: Juan Antonio Carral Pelayo

Vocal 2º: Elisa Rojas Sánchez

FECHA: 01/10/2021

Dedicatoria

A Oliver.

A todas las personas que me quieren y que quiero.

Contenido

Resumen

Programar redes definidas por software (en inglés Software-Defined Networking, SDN) es complejo, especialmente si no se conoce la arquitectura a bajo nivel del controlador y su interfaz de programación (en inglés, Application Prorgamming Interface, API). Actualmente existen numerosos trabajos de investigación que tratan de solventar este problema. En el presente Trabajo Fin de Grado se implementa una prueba de concepto de una interfaz visual con Blockly para la plataforma de SDN open source ONOS, en la que se maneja el tráfico en una topología triangular con los mensajes OpenFlow PacketIn y PacketOut.

Palabras clave

Redes definidas por software, SDN, ONOS, Blockly.

Abstract

Programming software-defined networking is complicated, especially without knowing the low-level architecture of the controller and the Application Programming Interface (API). There are multiple research works and developments trying to solve this problem. In this project we have implemented a visual interface using Blockly for the open source SDN platform ONOS. This interface can control the traffic in a triangular topology using the OpenFlow messages PacketIn and PacketOut.

Key Words

Software-defined networking, SDN, ONOS, Blockly.

1. INTRODUCCIÓN

La abstracción se define como: “aislar un elemento de su contexto o del resto de los elementos que lo acompañan” [1]. En el contexto de la informática se suele decir que cada vez la programación van avanzando a un nivel mayor de abstracción, lo que quiere decir que cada vez hace falta menos conocimientos avanzados para poder implementar una idea. Para explicar más en detalle este concepto, vamos a ver un ejemplo de un pequeño programa que suma dos números en dos lenguajes de programación, uno de bajo nivel de abstracción y uno de alto nivel.

Lenguaje de bajo nivel: (ensamblador x8086):

```
MOV AL, 5
SUM AL, 7
MOV RESULTADO, AL
```

Figura 1 Ejemplo de código ensamblador

Lenguaje de alto nivel (Python):

```
resultado=5+7
```

Figura 2 Ejemplo de código Python

Una diferencia que se puede observar comparando estos ejemplos es la de las direcciones de memoria. En el primer caso debemos mover los números a unas direcciones concretas de la memoria (el programador debe de estar pendiente del hardware), por ejemplo, si en la suma hay overflow (el resultado no puede ser representado con la capacidad de una variable y sobrepasa la misma), habrá un flag (una dirección de memoria) que pasará de 0 a 1. Podemos observar cómo en el lenguaje de alto nivel esto no ocurre, el hardware está abstraído, haciendo el desarrollo más sencillo.

Los lenguajes de bajo nivel además solo funcionan sobre el hardware para el que están desarrollados, por lo tanto, si cambiamos el hardware, debemos adaptar el software al set de instrucciones del nuevo hardware para poder desarrollar, no existe compatibilidad. Los lenguajes de alto nivel pueden correr independientemente del hardware que tengan debajo.

En el contexto de los lenguajes de programación, la abstracción ha avanzado muy rápidamente. En el apartado del estado del arte veremos la herramienta Blockly, un lenguaje de programación que nos permite programar con piezas de puzle, una manera muy visual que está orientado a un público infantil, este lenguaje llega a abstraer incluso la propia gramática de la programación.

No obstante, en el apartado de las redes de comunicación no se ha avanzado tanto como lo han hecho los lenguajes de programación, la mayoría de los despliegues aún se hacen con hardware propietario.

En la última década (2010) surgen las redes definidas por software (del inglés, Software-Defined Networking, SDN) [2][3]. El desarrollo de estas permite separar el hardware (plano de datos) del software (plano de control). Esto tiene grandes implicaciones, la principal y más importante es que podemos desarrollar sin necesidad de conocer en profundidad el hardware, existe compatibilidad entre diferentes dispositivos independientemente de la empresa que lo fabrique. Además, proporciona un nivel extra de abstracción a la hora de desarrollar protocolos y servicios de red, ya que, podemos tomar la red como un todo en vez de tener que configurar de manera individual cada dispositivo.

Las SDN han avanzado mucho en los últimos años permitiendo por ejemplo desarrollar con lenguajes de alto nivel como Python (Ryu) o Java (ONOS, del que hablaremos más adelante y utilizaremos en este trabajo). No obstante, desarrollar redes definidas por software sigue siendo un reto especialmente si no tenemos conocimiento en redes.

El objetivo de este trabajo es el de añadir un nivel de abstracción a las SDN, diseñando e implementando una interfaz visual con Blockly a la plataforma de redes definidas por software de ONOS.

Este trabajo es una ampliación del trabajo fin de grado “Estudio de interfaz de control simplificado para plataforma SDN” de la alumna Victoria Noci Luna, en el cual se realiza un primer acercamiento a la idea de aplicar un entorno de abstracción de programación a las redes SDN.

1.1 ESTRUCTURA DE LA MEMORIA

Una vez vista esta pequeña introducción, en el próximo apartado, explicaremos qué son las redes definidas por software, repasaremos las diferentes herramientas que vamos a utilizar para el desarrollo del trabajo y también expondremos algunas de las investigaciones que se están haciendo en relación al presente Trabajo Fin de Grado (TFG).

Después de esto, procederemos al diseño del proyecto, un breve apartado donde se abordará qué es lo que vamos a hacer, cuál es la motivación del proyecto, así como por qué se han elegido estas herramientas y una comparativa de éstas con algunas similares.

En el penúltimo apartado, pasaremos a la implementación donde se irá paso a paso, desde la instalación de las herramientas hasta la finalización de todo el proyecto, pasando por todo el desarrollo del código necesario, con imágenes de todo el proceso.

Finalmente pasaremos a la evaluación, donde con un caso de uso se demostrará el correcto funcionamiento de la implementación y terminamos con las conclusiones de

todo el trabajo realizado, así como con un conjunto de ideas para el trabajo futuro de la investigación.

2. ESTADO DEL ARTE

En este segundo apartado expondremos inicialmente los conceptos de SDN y OpenFlow, claves para poder entender la motivación de este trabajo. Tras esto, haremos un repaso a las herramientas a utilizar: ONOS, Mininet y Blockly. Finalmente, haremos una pequeña exposición de otras investigaciones en el campo de las SDN, como son P4, Lucid y Graph to P4.

2.1 SDN

Las redes definidas por software son un conjunto de técnicas que facilitan la implementación de servicios de red, evitando al administrador de la red gestionar los servicios de bajo nivel.

Esto se consigue separando el plano de control del plano de datos, como se puede ver en la Figura 3. En una red tradicional, en cada switch se manejan ambos planos, mientras que en una arquitectura SDN, el plano de control se separa, dejando a los switches tan solo manejar el plano de datos.

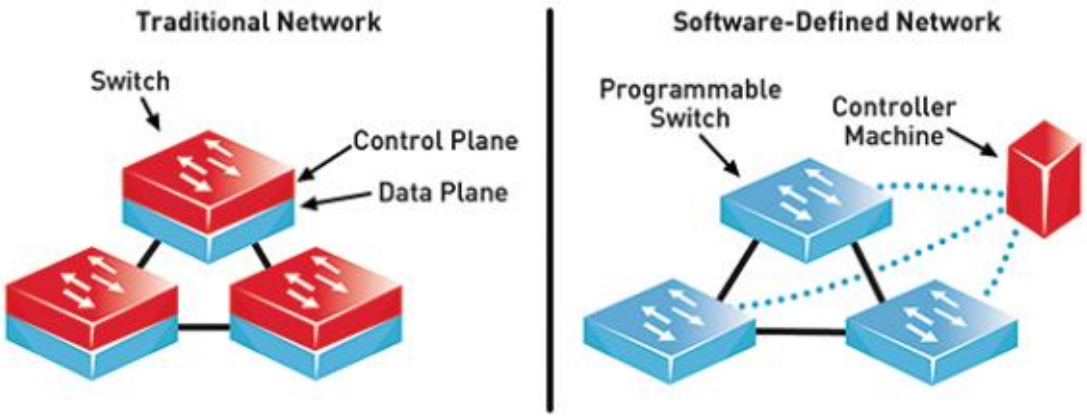


Figura 3 Comparativa SDN - Red tradicional

Este avance tiene numerosas ventajas:

- Capacidad para manejar los recursos de manera dinámica.

- Mayor agilidad en el despliegue y control de las redes.
- Gestión centralizada.
- Seguridad.
- Monitorización.

2.1.1 OPENFLOW

OpenFlow es una tecnología de switching originada en 2008, es un protocolo que permite a un servidor de software determinar el camino de reenvío de paquetes que debería seguir en una red de switches [4].

En una red sin OpenFlow cada switch tiene un software que le dice qué hacer, la transferencia de paquetes (es decir el plano de datos) y el enrutamiento de alto nivel (plano de control) ocurre en el mismo dispositivo. Por otro lado, en una red con OpenFlow se separa el plano de datos y el plano de control: mientras el switch se seguirá encargando de la transferencia de paquetes tenemos el controlador que se encarga del enrutamiento de alto nivel y estos dos se comunican mediante el protocolo OpenFlow [5].

Está metodología es conocida como redes definidas por software de la cual ya hemos hablado en el apartado anterior y permite un uso más efectivo de los recursos de la red además de permitir que la red se pueda gestionar de manera completa en vez de tener que gestionar los dispositivos de manera individual.

Los switches OpenFlow cuentan con tres partes:

- Tabla de flujos: es una tabla donde se asocia una acción a cada entrada de la tabla indicando al switch como actuar, por ejemplo “Si llega un paquete desde X” -> “mandar a Y”, en caso de que llegara un paquete para el cual no hay una entrada en la entrada de flujos, se reenviará al controlador que decidirá como actuar y añadirá la entrada en la tabla de flujos para el futuro.
- Conexión mediante protocolo OpenFlow entre el switch y el controlador.
- Controlador: añade, elimina o modifica entradas de la tabla de flujos.

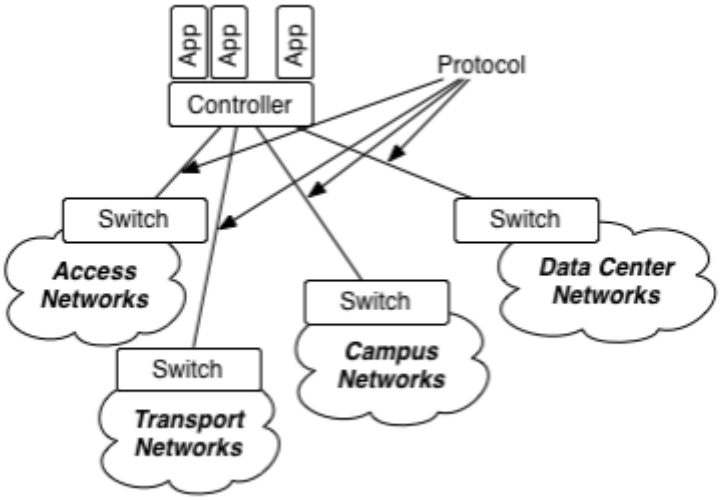


Figura 4 Diagrama OpenFlow

Todos los mensajes OpenFlow tienen la misma estructura inicial donde se indica la versión de OpenFlow, el tipo de mensaje que es, la longitud, indicando cual es el último byte y el xid, que es un indicador de transacción, con un valor único para identificar que respuestas van con que solicitud.

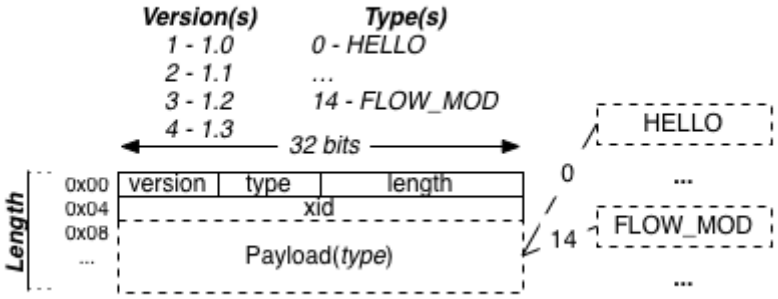


Figura 5 Estructura de mensajes OpenFlow

Hay muchos tipos de mensajes OpenFlow, pero en este trabajo utilizaremos dos de ellos [6]:

PacketIn

El mensaje de PacketIn es un mensaje que manda el switch al controlador. Esto puede ocurrir cuando no hay entrada en la tabla de flujo para esa entrada o cuando en la tabla de flujos dice que esa entrada debe resolverse con un PacketIn.

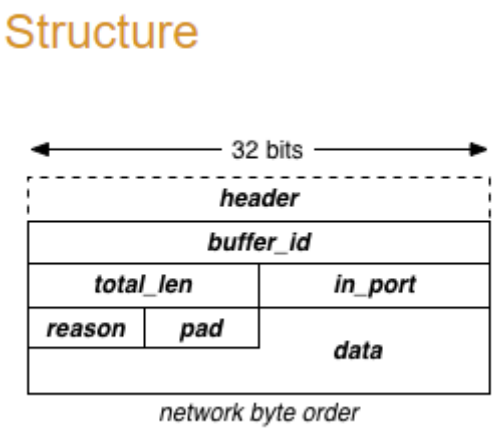


Figura 6 Estructura del mensaje PacketIn

PacketOut

El mensaje de PacketOut es un paquete inyectado por el controlador en el switch. Los paquetes inyectados de esta manera no son tratados como un paquete normal que llega al switch si no que son mandados directamente a ser procesados.

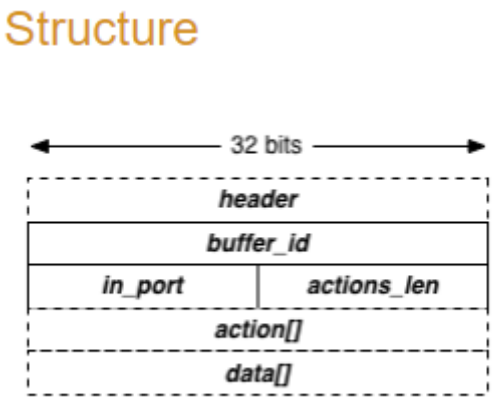


Figura 7 Estructura del mensaje PacketOut

2.1.2 P4

P4 (Programming Protocol-independent Packet Processors) es un lenguaje de programación open source específico para programar el plano de datos de dispositivos de red [7].

El proyecto P4 comienza en 2014 donde se presenta como el futuro de las redes SDN, hoy en día cuenta con una comunidad enorme y con un gran desarrollo desde su lanzamiento, se presenta para resolver los problemas del desarrollo de una red de alto rendimiento, donde anterior a P4 era necesario un largo proceso de selección de hardware, firmar un acuerdo para conseguir un SDK y tener que aprender todo ese kit de desarrollo de 0 [8].

Con P4 potencialmente se obtienen los beneficios de la ingeniería de software, como pueden ser el debugging, model checking o el desarrollo independientemente del hardware que utilicemos.

2.1.3 MININET

Mininet es un emulador de redes SDN de código abierto y fácilmente configurable que genera una red con hosts virtuales, switches OpenFlow, controladores y enlaces [9].

Mininet es la plataforma de apoyo a la investigación de las SDN más popular. Permite el uso de topologías de red completamente customizables que se pueden crear de manera muy sencilla con un pequeño archivo en Python, aunque también tiene varias topologías ya disponibles sin tener que programar nada, esto es de gran utilidad a la hora de hacer pruebas. Por ejemplo, para el desarrollo de este trabajo se estuvo probando con topologías ya incluidas en Mininet, aunque para las pruebas finales utilizamos una propia.

Una de las características más útiles de esta herramienta para el desarrollo de este trabajo es el CLI (Command Line Interface) que nos permite realizar diferentes acciones en tiempo de ejecución, ya hemos hablado de que está desarrollado en Python, pero además su consola puede ejecutar diferentes comandos con Python, como, por ejemplo:

```
mininet> py locals()
```

Figura 8 Comando py locals() de Mininet

Comando que nos permitirá ver las variables locales.

Figura 9 Comando py dir(s1) de Mininet

Que nos permitirá ver métodos y propiedades disponibles para el nodo que hemos indicado (en este caso el switch 1).

Por otro lado, podemos realizar acciones como desactivar o activar un enlace entre dos dispositivos con el comando:

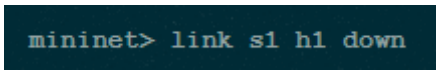


Figura 10 Comando link s1 h1 down de Mininet

Figura 11 Comando link s1 h1 up de Mininet

O acceder a una miniconsola Xterm para un dispositivo:

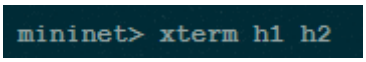


Figura 12 Comando xterm h1 h2 de Mininet

2.1.4 ONOS

ONOS (Open Network Operating System) es un proyecto que comenzó en 2012 por Pankaj Berde, fue mostrado en abril de 2013 en el Open Networking Summit y las primeras versiones salieron en 2014, en este año fue cuando se liberó el código para que el proyecto fuera Open Source [10][11].

ONOS es un controlador de redes definidas por software (en inglés Software-Defined Networking, SDN) que permite crear al usuario de manera sencilla desarrollar aplicaciones con lenguajes de alto nivel (Java) sin tener que conocer el plano de datos.

En este trabajó utilizaremos Mininet para desplegar una red, eligiendo la topología, es decir, el número de switches, de hosts y todos los enlaces entre ellos y lo conectaremos a ONOS, lo cual nos permitirá administrar y configurar la red, así como desarrollar y desplegar software sobre está red.

ONOS se lanza con el siguiente comando, donde indicamos las opciones de clean para que limpie la instalación, y el comando debug que da la opción de que el puerto 5005 este libre para adjuntar un debugger de Java remoto:

```
alvaro@tfg:~/onos$ bazel run onos-local -- clean debug
```

Figura 13 Comando de lanzamiento de ONOS

El comando tarda un rato en ejecutarse por completo, lo tendremos listo cuando aparezca como en la siguiente figura la IP y el estado en READY.

```
alvaro@tfg: ~/onos
File  Edit  View  Search  Terminal  Help
                | 106 - org.ops4j.pax.web.pax-web-jsp - 7.2.16 | found TLD bundl
e://106.0:1/META-INF/x-1_0.tld
2021-08-23T00:40:50,379 | INFO  | paxweb-extender-2-thread-1 | TldScanner
                | 106 - org.ops4j.pax.web.pax-web-jsp - 7.2.16 | found TLD bundl
e://106.0:1/META-INF/x.tld
2021-08-23T00:40:50,529 | INFO  | SCR Component Actor | DistributedMeterStore
        | 192 - org.onosproject.onos-core-dist - 2.6.0.SNAPSHOT | Started
2021-08-23T00:40:50,538 | INFO  | SCR Component Actor | MeterManager
        | 193 - org.onosproject.onos-core-net - 2.6.0.SNAPSHOT | Configured. Pu
rgeOnDisconnection is disabled
2021-08-23T00:40:50,544 | INFO  | SCR Component Actor | MeterManager
        | 193 - org.onosproject.onos-core-net - 2.6.0.SNAPSHOT | Started
2021-08-23T00:40:50,599 | INFO  | paxweb-extender-2-thread-1 | ContextHandler
                | 80 - org.eclipse.jetty.util - 9.4.28.v20200408 | Started HttpS
erviceContext{httpContext=WebAppHttpContext{org.onosproject._onos-gui2-base-jar
- 201}}
2021-08-23T00:40:50,789 | INFO  | CM Event Dispatcher (Fire ConfigurationEvent:
pid=org.onosproject.net.meter.impl.MeterManager) | MeterManager
    | 193 - org.onosproject.onos-core-net - 2.6.0.SNAPSHOT | Configured. PurgeOn
Disconnection is disabled
2021-08-23T00:40:52,950 | INFO  | atomix-cluster-events | AtomixClusterStore
          | 195 - org.onosproject.onos-core-primitives - 2.6.0.SNAPSHOT | Updat
ed node 127.0.0.1 state to READY
```

Figura 14 Resultado del lanzamiento de ONOS

Una vez que ONOS está listo podemos ir a otro terminal y ejecutar el siguiente comando:

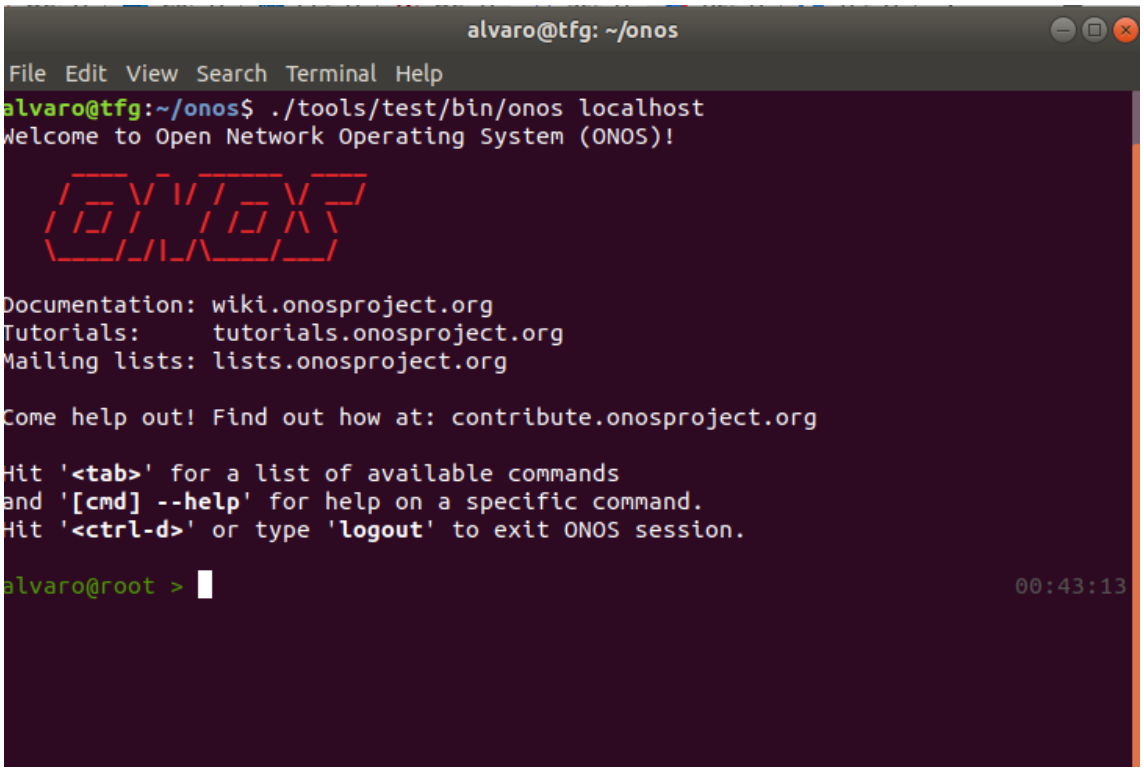


Figura 15 Consola de ONOS

Lo cual nos deja en este terminal una conexión con el servidor donde podemos realizar múltiples actividades: activar y desactivar aplicaciones, añadir reglas de flujo, añadir un nuevo cluster…

También ONOS cuenta con una interfaz web, para acceder a esta, debemos abrir cualquier navegador e ir a la dirección:

http://localhost:8181/onos/ui/login.html

Es importante destacar que todas las acciones a realizar como las mencionadas anteriormente se deben de hacer por consola (la mostrada en la figura 15), la interfaz web solo nos presenta la información de una manera más visual.

Esto nos lleva a una página web donde podemos introducir el usuario y contraseña, que por defecto son:

User: karaf

Password: karaf

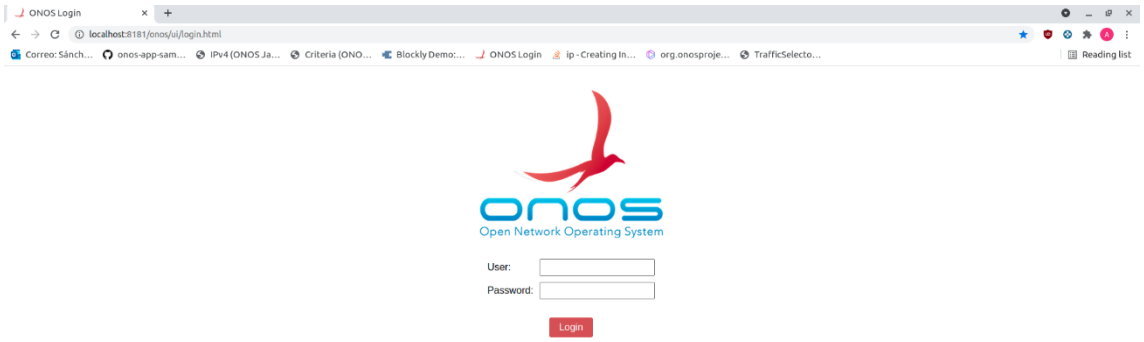


Figura 16 Pantalla de inicio de sesión ONOS

Esto nos llevará a poder observar toda la información de nuestro servidor de una manera mucho más sencilla.

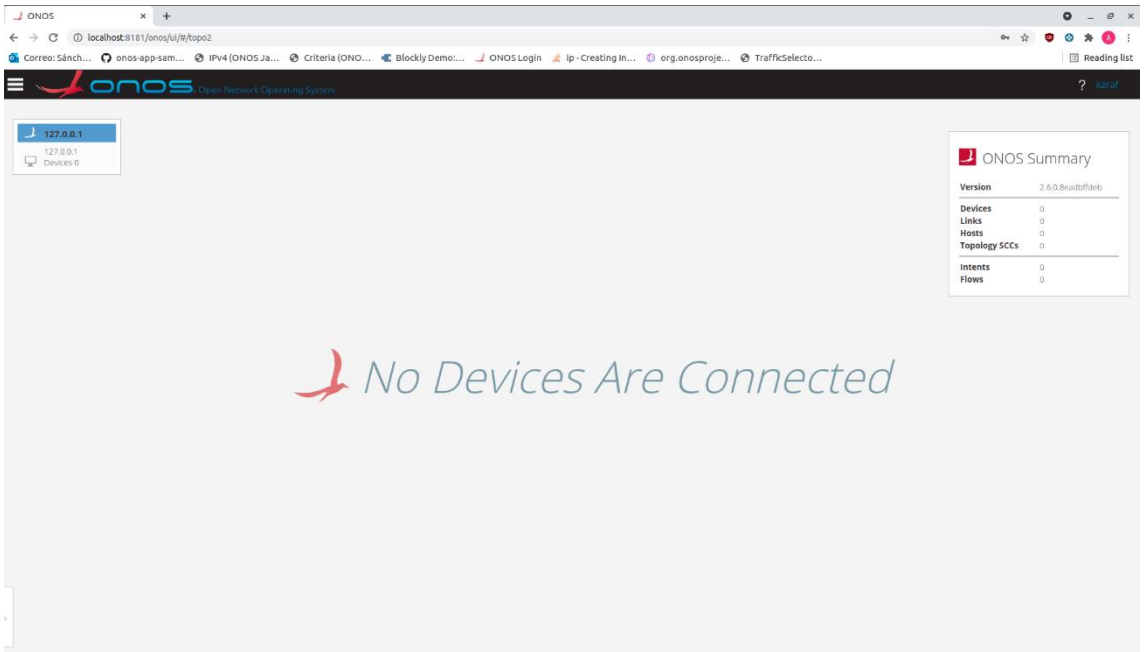


Figura 17 Interfaz web ONOS

En este caso al no haber ninguna red desplegada no se ve ningún dispositivo en la figura anterior, pero a continuación vamos a desplegar con Mininet una red:

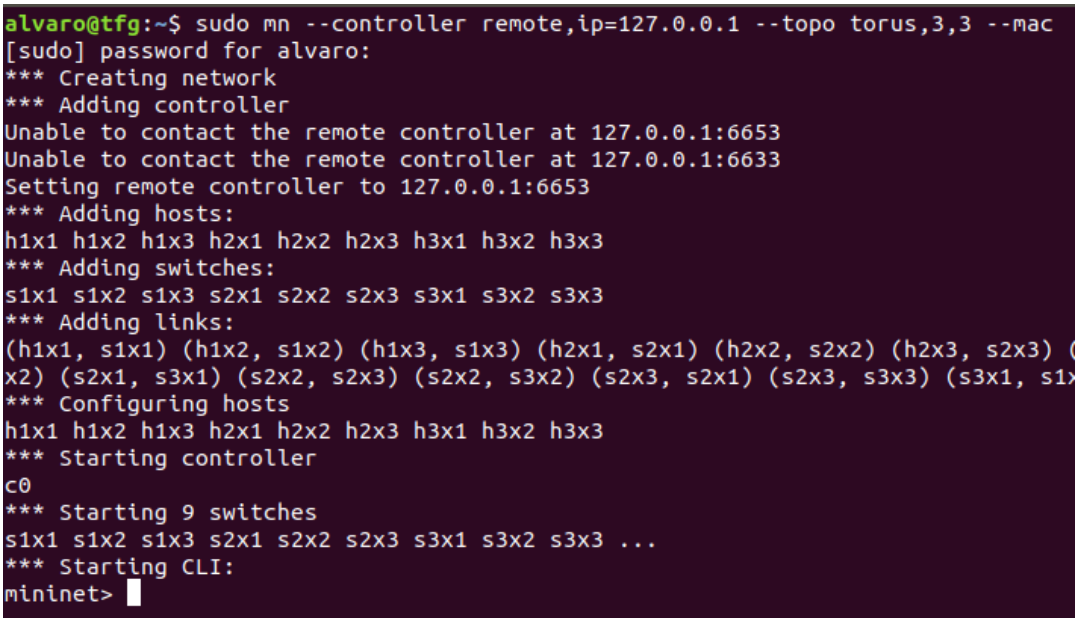


Figura 18 Lanzamiento de Mininet

Por defecto en ONOS no hay flujos instalados en el plano de datos por lo que, para que los paquetes puedan moverse por la red que hemos creado con Mininet, es importante activar dos aplicaciones: OpenFlow y Forwarding.

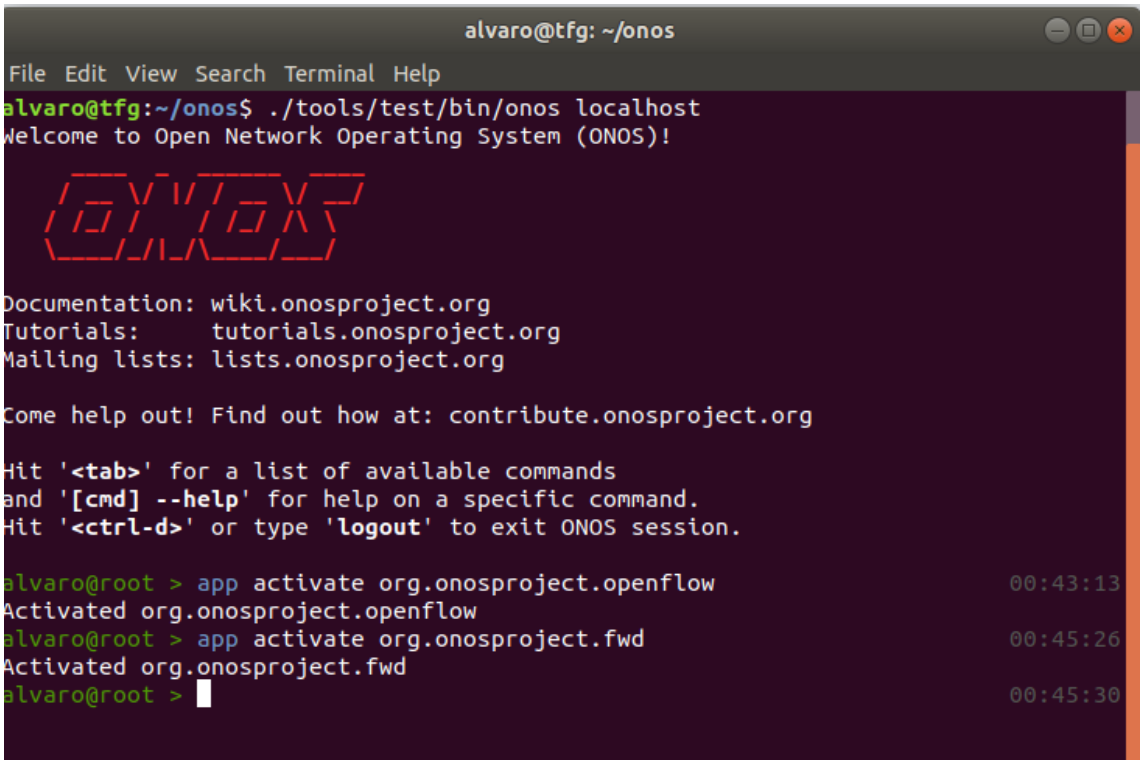


Figura 19 Activación de openflow y fwd en ONOS

Tras haber realizado estos pasos debería haber conexión entre todos los dispositivos, podemos comprobarlo con el comando “pingall” en la consola de Mininet, que como su propio nombre indica realiza un ping entre todos los pares de dispositivos:

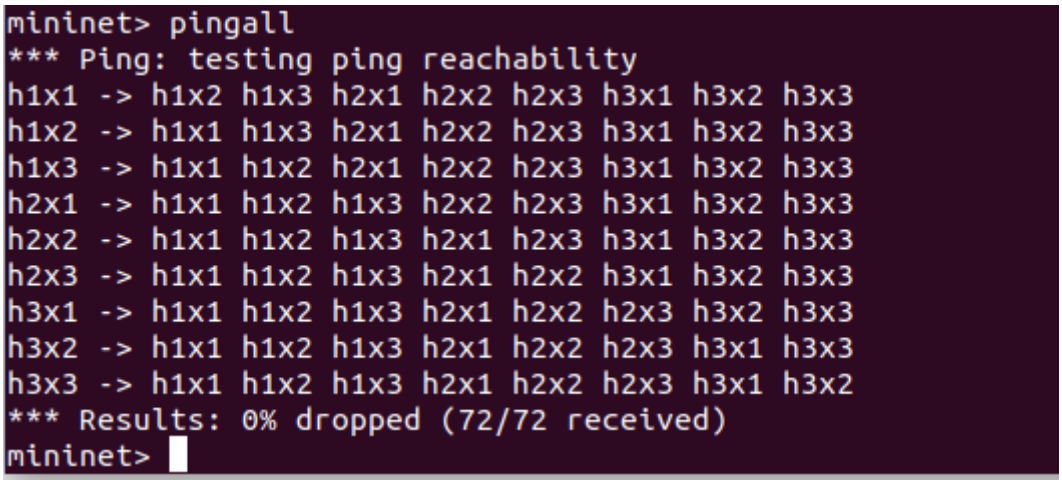


Figura 20 Muestra de pingall en Mininet

Podemos ver como todos los paquetes llegan a su destino, ahora vamos a observar también la interfaz web que hemos enseñado anteriormente. En concreto, en la página principal que antes indicaba que no teníamos dispositivos conectados nos aparece un

diagrama de la topología que estamos utilizando junto con un pequeño resumen en la parte superior derecha con alguna información general. Pulsando en la topología podemos acceder a información más concreta del dispositivo que seleccionemos.

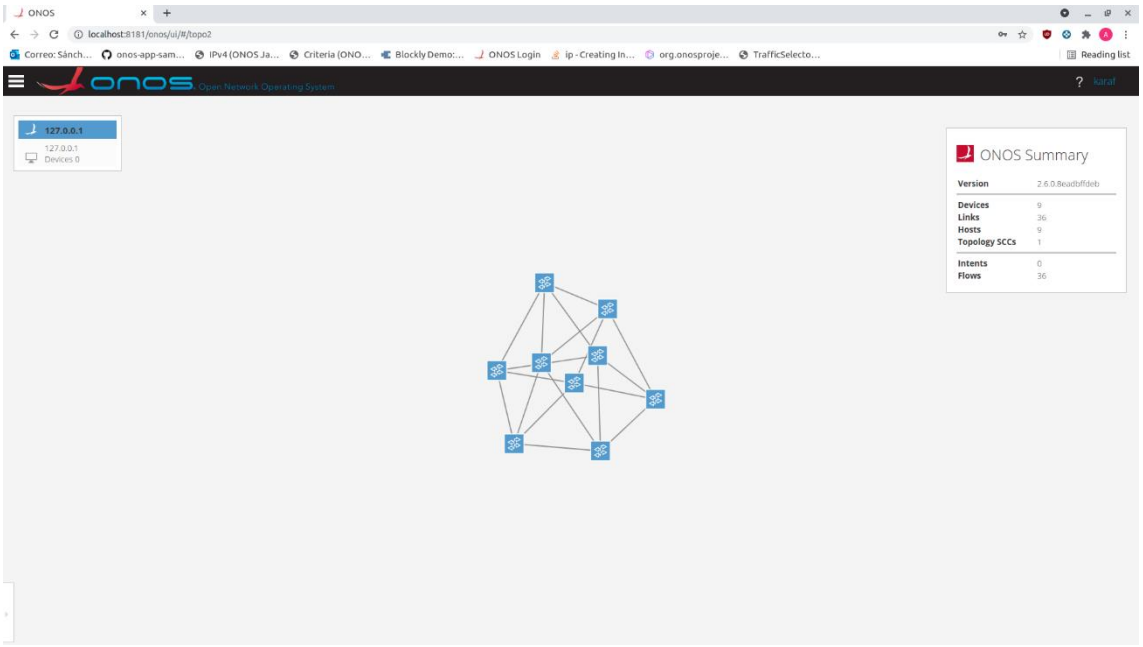


Figura 21 Interfaz web ONOS

También en esta pantalla es importante destacar la parte superior izquierda, donde tenemos un pequeño desplegable que nos mostrará la opción de ir moviéndonos por diferentes pestañas, entre ellas la de topología que era la que hemos visto en la figura 20, que es la predeterminada por defecto al entrar en la interfaz web. Vamos a ir viendo las más importantes:

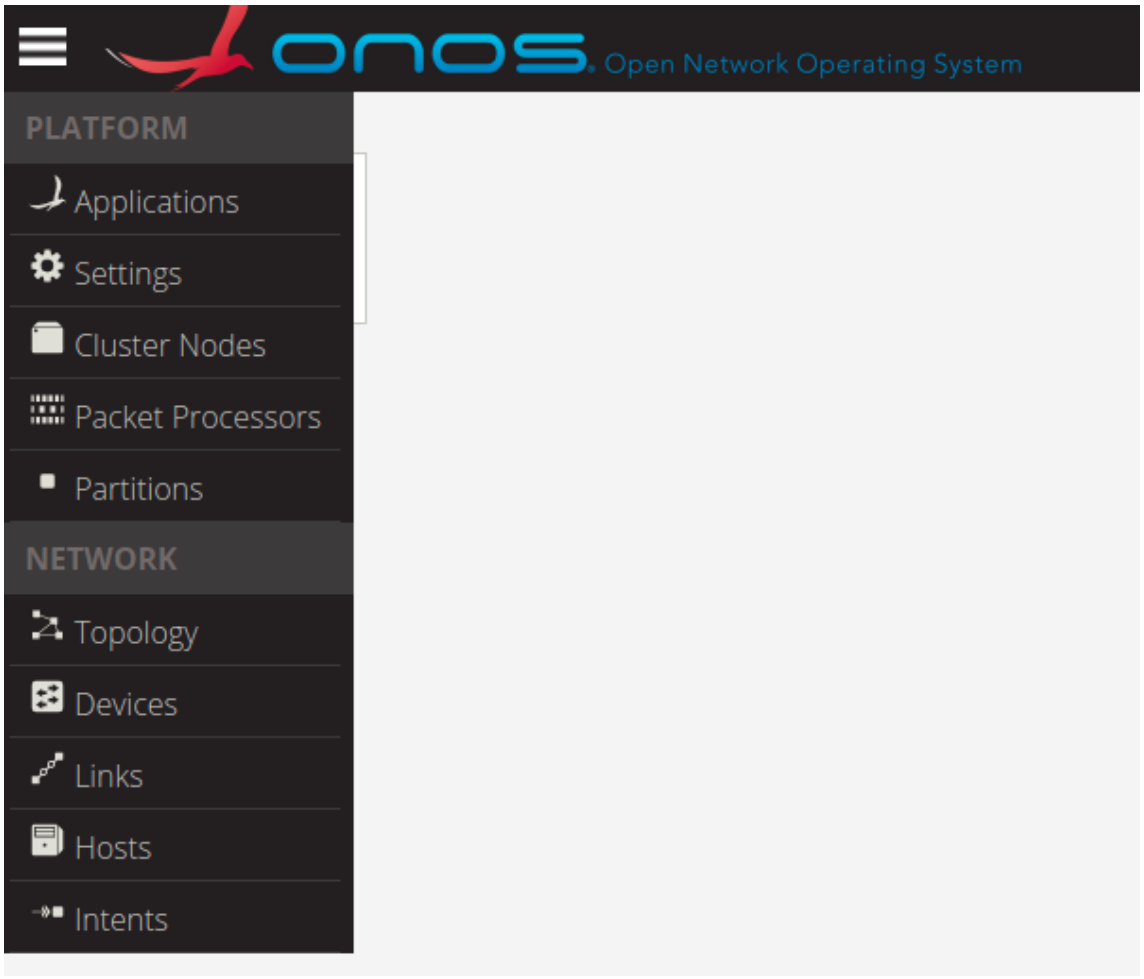


Figura 22 Interfaz web ONOS (desplegable)

En el apartado de aplicaciones se pueden consultar todas las aplicaciones instaladas y las que están activadas y no, luego en el desarrollo veremos cómo crear nuestra propia aplicación en ONOS que aparecerá aquí al instalarla.

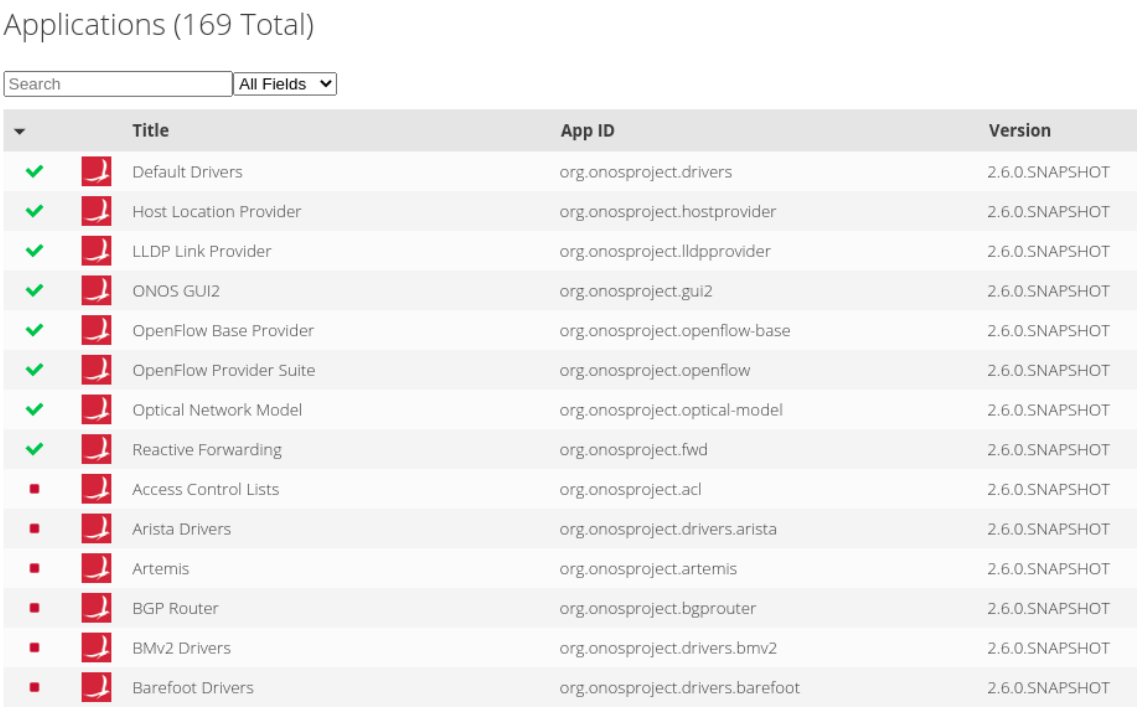


Figura 23 Interfaz web de ONOS (aplicaciones)

En el apartado de dispositivos se representan todos los dispositivos de la red, con sus puertos, sus direcciones MAC e IP, esto es de gran utilidad especialmente cuando no estemos en un escenario de pruebas donde las MAC y las IP estén listadas en orden (esto lo conseguimos con la opción –mac de Mininet) ya que cuando las MAC e IP son aleatorias es en este apartado donde tendremos que venir a consultarlas.

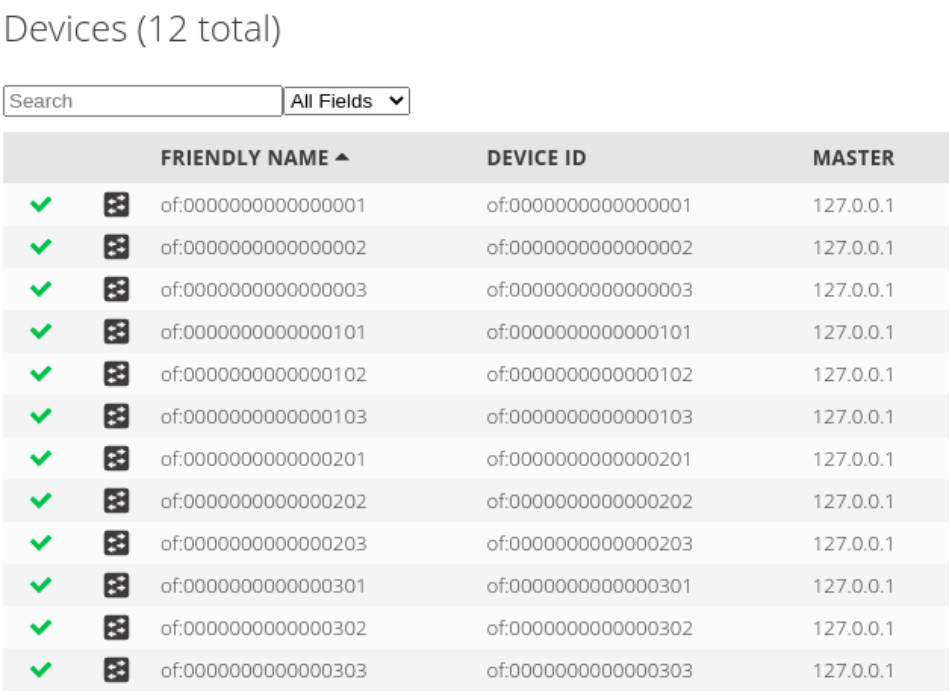


Figura 24 Interfaz web de ONOS (Dispositivos)

En el apartado de links se reflejan todos los enlaces entre los dispositivos que hemos visto en la figura anterior, también podemos observar si estos enlaces están activos o no y si es bidireccional o tan solo en una dirección.

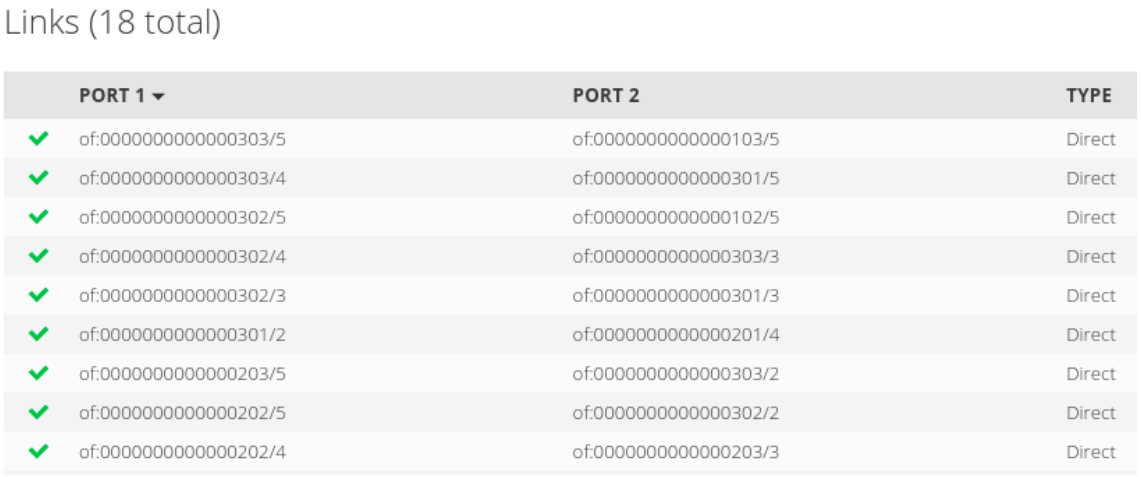


Figura 25 Interfaz web de ONOS (Enlaces)

Y en la última de las pestañas que tienen relevancia para este trabajo nos encontramos el apartado de Hosts, donde podemos ver la información relacionada con estos, como su ID, MAC o su ID VLAN.

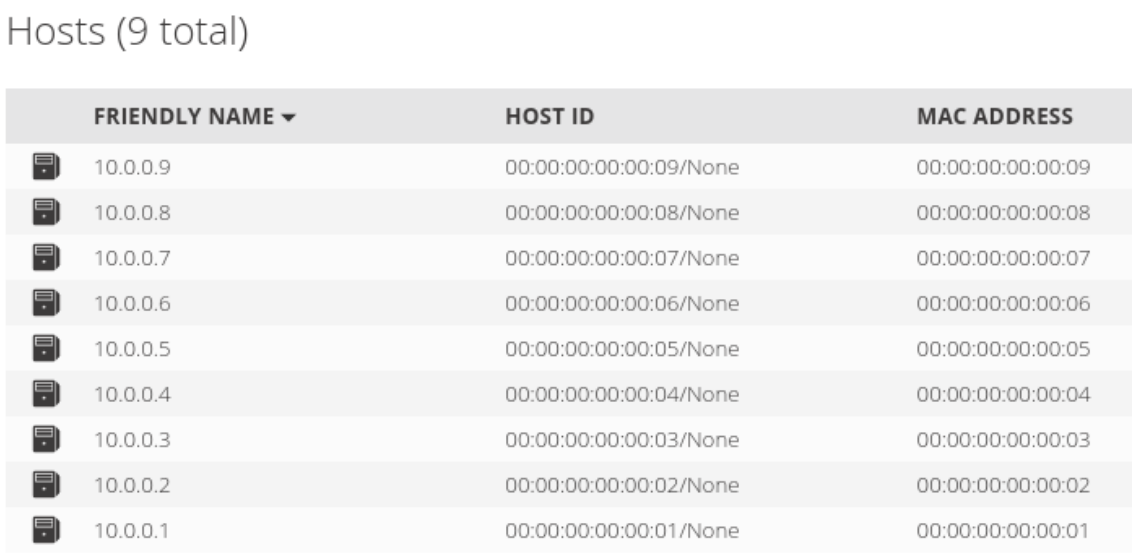


Figura 26 Interfaz web de ONOS (Hosts)

ONOS permite una interfaz de control de la red desplegada muy visual y sencilla de manejar, veremos en el apartado de desarrollo como instalar ONOS y como desarrollar aplicaciones para este framework, no es una herramienta especialmente sencilla de utilizar, pero tiene múltiples de características muy útiles para el despliegue de redes como hemos podido observar.

2.2 PROGRAMACIÓN EN BLOQUE

2.2.1 BLOCKLY

Blockly es un lenguaje de programación visual, es un proyecto de Google y es un software libre y de código abierto. Blockly utiliza bloques que se conectan unos con otros a modo de puzle y permitiendo traducir el código que estamos creando en este entorno visual a código JavaScript, Lua, Dart, Python o PHP [12] [13].

El entorno visual permite abstraer la programación un nivel más que los lenguajes de alto nivel, ya que, con la programación visual, no es necesario conocer la gramática del propio lenguaje lo que simplifica en gran medida la tarea de programar.

Este tipo de programación está orientada a un público que desea introducirse al mundo de la programación, especialmente niños, de hecho, vemos como en la página principal de Blockly nos enseñan una serie de juegos para aprender a programar destinados a este tipo de público, no obstante, es un lenguaje muy potente con el que además podemos programar lo que queramos con una interfaz muy sencilla.

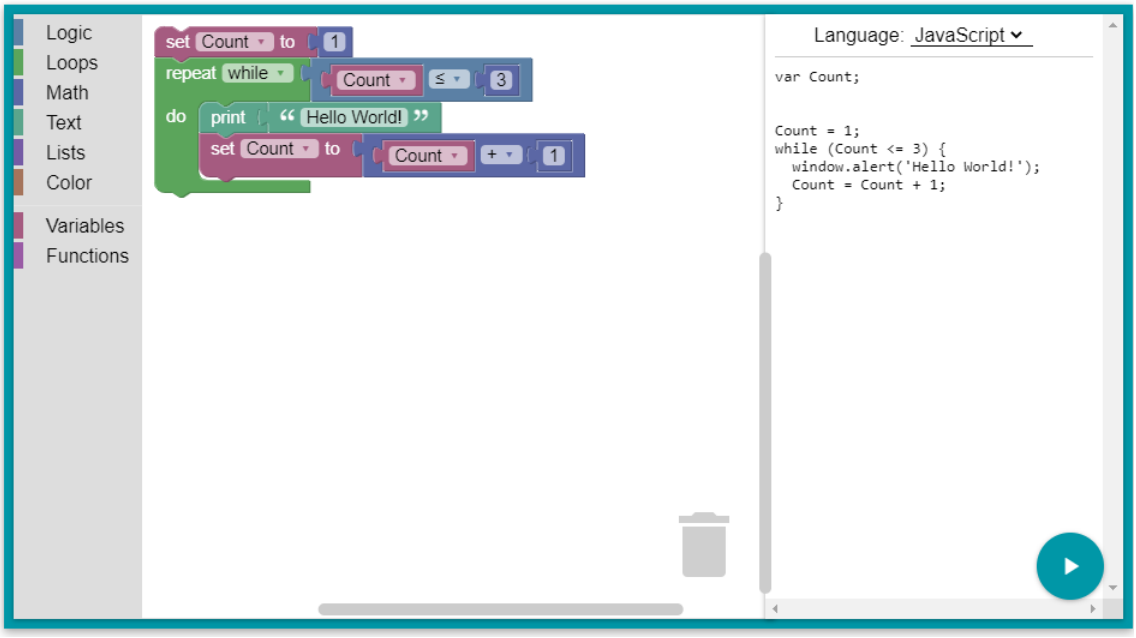


Figura 27 Ejemplo de Blockly

En la Figura 26 se muestra la demo de “Code”, una interfaz donde podemos elegir los bloques lógicos de los lenguajes de programación de alto nivel y encajarlos entre ellos mientras que en el apartado de la derecha se nos traduce esos bloques al lenguaje que elijamos entre los disponibles.

Está demo es importante porque en este trabajo partiremos de está demo, creando nuestros propios bloques y dándoles un código a estos bloques personalizados para que cuando los juntemos entre ellos se nos cree un código Java que podamos exportar para crear una aplicación ONOS que maneje la red desplegada con Mininet.

Para crear nuestros propios bloques debemos de ir a las herramientas de desarrollador de Blockly donde tenemos una herramienta para hacerlo:

Figura 28 Blockly Developer Tools

Como se puede observar de una manera muy clara, podemos elegir en la parte de la izquierda las entradas, los campos, los tipos…, los cuales se juntan en la parte central creando un bloque tal como queremos, por último, en la parte de la derecha tenemos el código que este bloque genera, en el caso de este trabajó partimos de los bloques creados por Victoria en su trabajó de fin de grado.

2.3 PROGRAMACIÓN DE ALTO NIVEL EN REDES SDN

En este último apartado del Estado del Arte, veremos que otras investigaciones se están realizando en el área de las redes definidas por software. Comenzando por el Trabajo Fin de Grado de Victoria Noci, del que nace este proyecto. Luego expondremos Lucid, un lenguaje de programación nuevo para SDN. Finalmente veremos un pequeño toolset que permite crear grafos y que éstos se conviertan a código P4 automáticamente.

2.3.1 TRABAJO DE FIN DE GRADO DE VICTORIA NOCI LUNA

Ya hemos mencionado en la introducción que este proyecto parte de la investigación que hizo la alumna Victoria Noci Luna en su trabajó de fin de grado “Estudio de interfaz de control simplificado para plataforma SDN” [14].

En este trabajo Victoria utiliza Mininet para crear una red personalizada en forma triangular con el siguiente esquema:

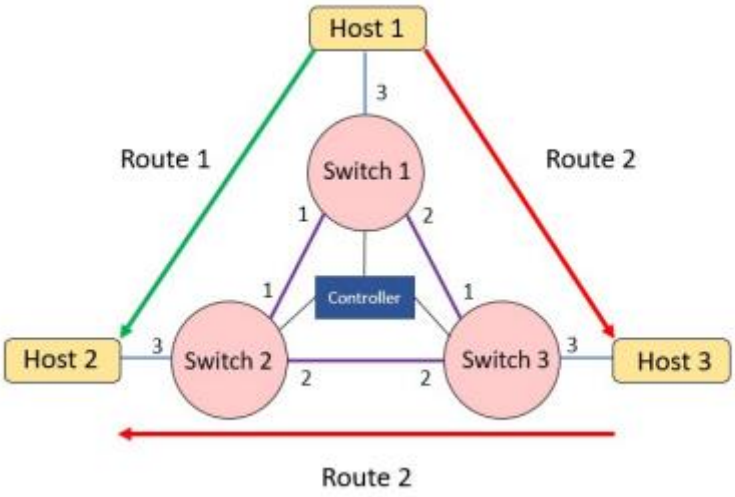


Figura 29 Topología del trabajo de Victoria

Posteriormente utiliza la herramienta de Ryu, un framework SDN desarrollado en Python que soporta el protocolo OpenFlow, es similar a ONOS del cual ya hemos hablado, aunque más sencillo.

En Ryu, Victoria modifica un controlador para que muestre la información de un PacketIn cuando llega a un switch.

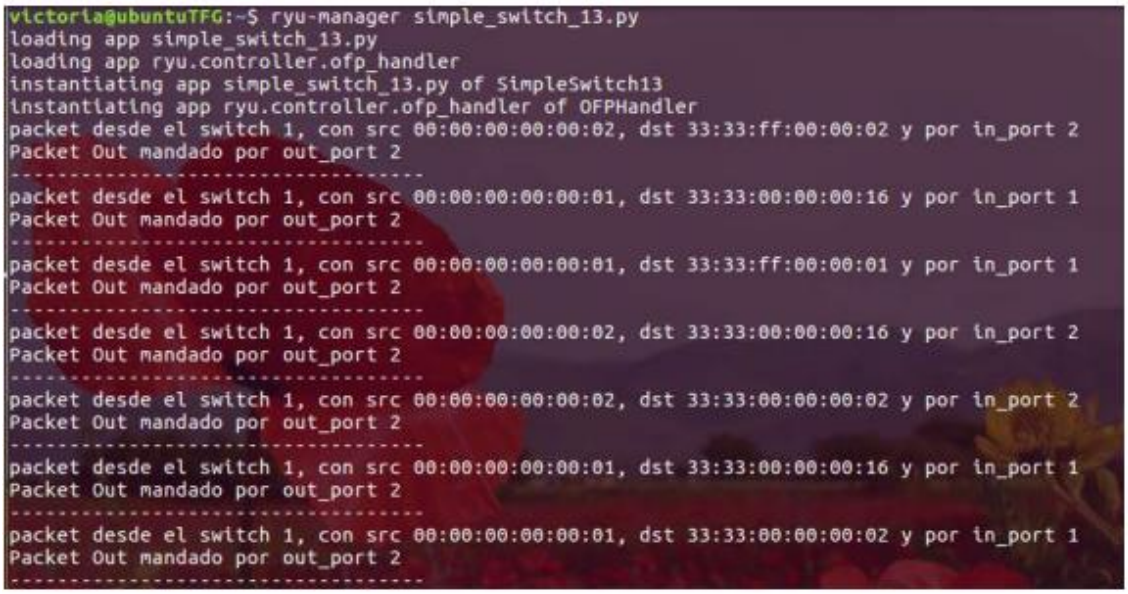


Figura 30 Demostración del trabajo de Victoria

Por el lado de Blockly, Victoria modifica la demo “Code” de Blockly incluyendo un nuevo apartado de bloques “Handlers” donde introduce los bloques que crea con las Developers Tools de Blockly [15].

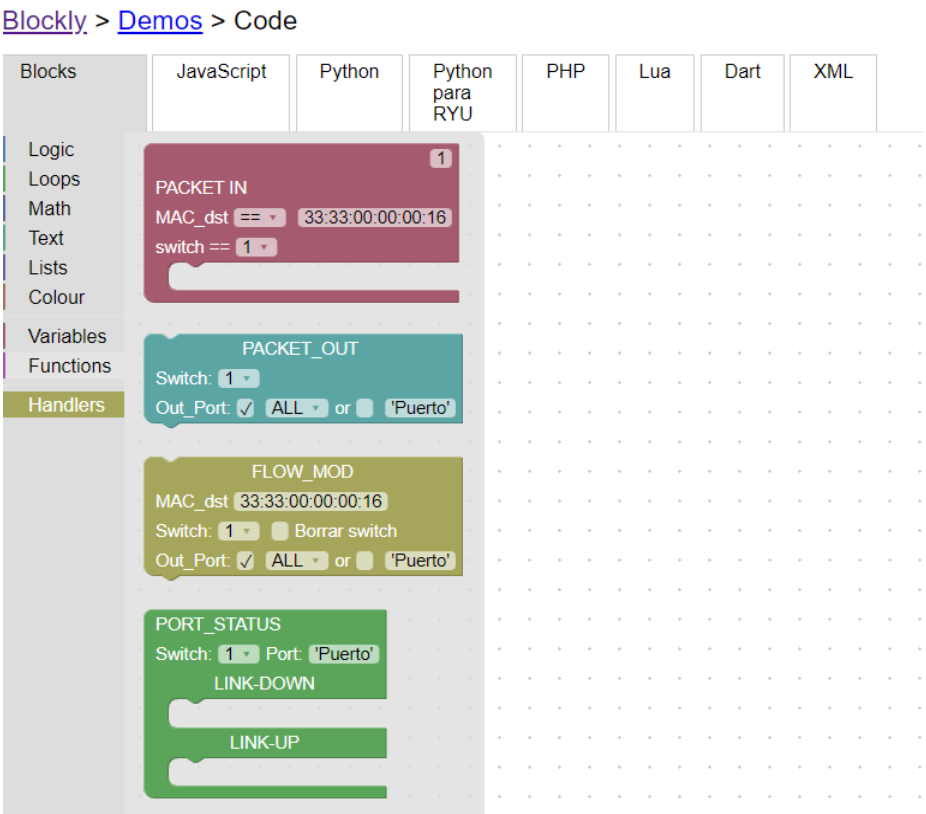


Figura 31 Muestra de los bloques de Victoria

También introduce un apartado de “Python para Ryu” donde al poner los bloques crean un código Python compatible con Ryu que simulan los mensajes OpenFlow correspondientes.

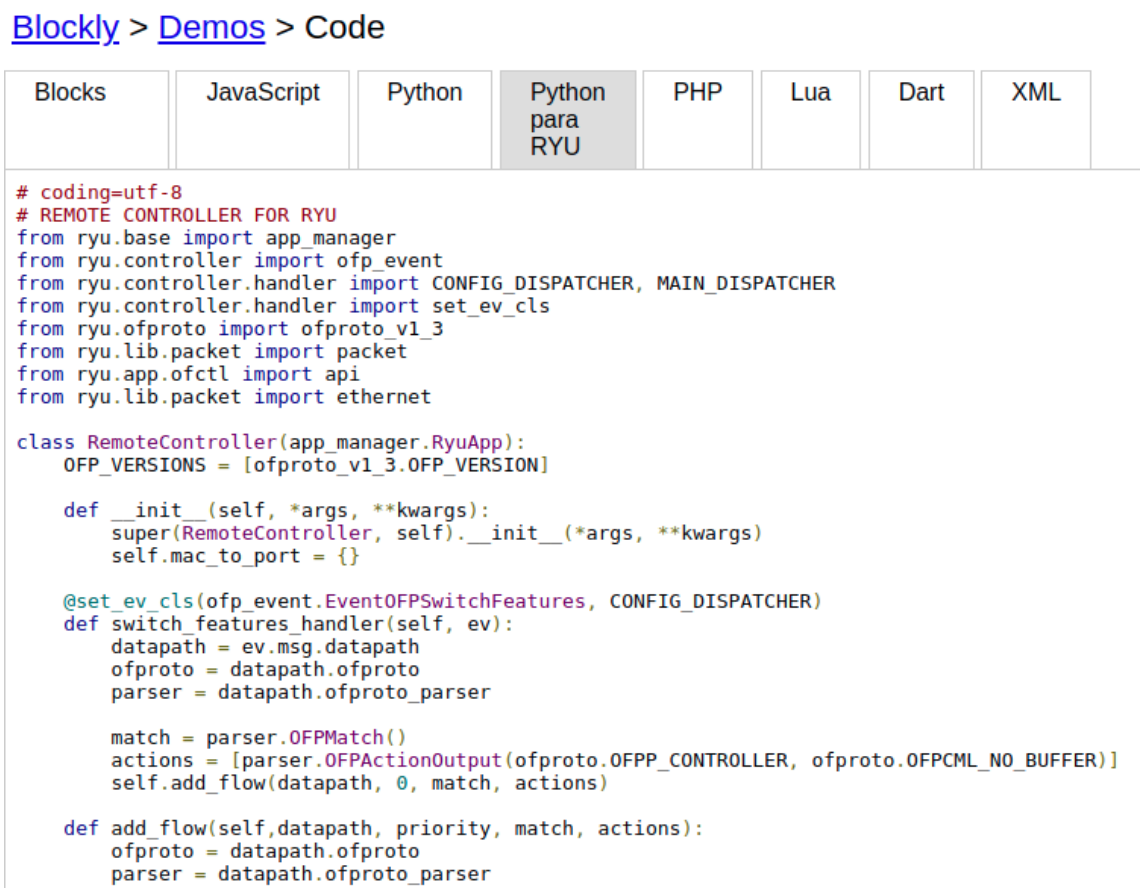


Figura 32 Ejemplo código de Victoria

Este código creado se exporta a Ryu donde simulará el comportamiento de los mensajes OpenFlow. Por ejemplo, la Figura 32 muestra como un PacketIn hace que los switches añadan reglas en la tabla de flujos.

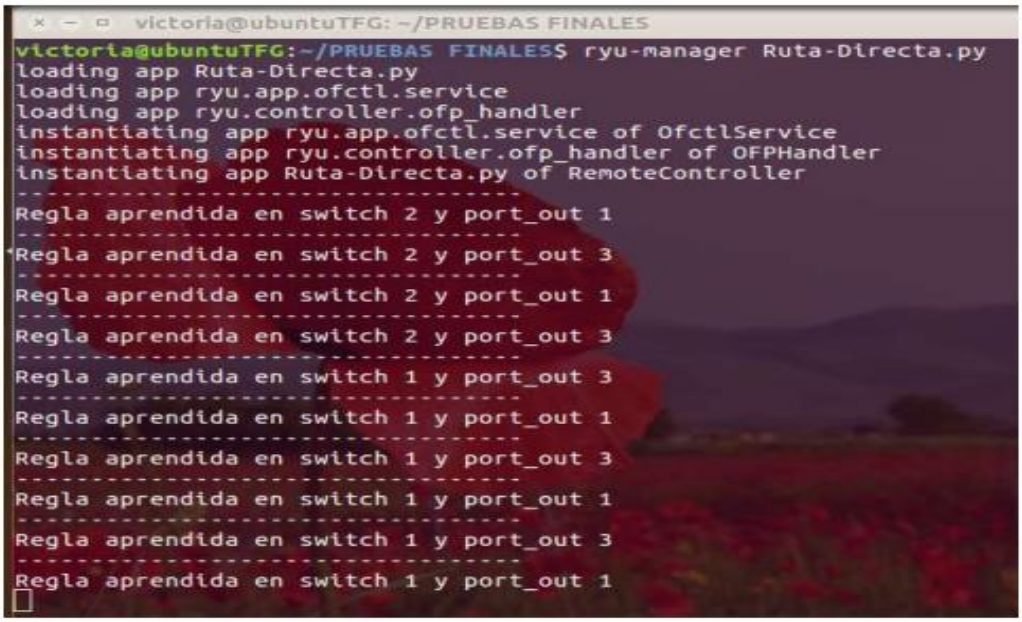


Figura 33 Ejemplo del trabajo de Victoria

2.3.2 LUCID

Ya hemos hablado en la parte de SDN de P4 y de cómo trata de resolver los problemas de las redes SDN, no obstante, P4 tampoco es un lenguaje sencillo. Es por ello por lo que también se están intentando crear nuevos lenguajes que sean más sencillos de utilizar que este, por ejemplo, Lucid, presentado en agosto de 2021 por John Sonchack, Devon Loehr, Jennifer Rexford y David Walker [16].

Lucid es un lenguaje de programación de alto nivel para implementar aplicaciones de control para arquitecturas PISA, pudiendo desplegar estas aplicaciones en un sólo switch de la red o en varios de ellos. Lucid permite a los programadores crear programas de alto nivel de abstracción para manejar. Comparado con P4, los programas de Lucid utilizan entre 5 y 10 veces menos líneas de código que su equivalente en P4, además de ser más eficiente en la compilación.

```
Array nexthops = new Array<<32>>(NUM_HOSTS);
Array pcts = new Array<<32>>(NUM_PORTS_X3);
Array hcts = new Array<<32>>(NUM_HOSTS);
memop plus(int cur, int x){return cur + x;}

event count_pkt(int dst, int proto);
handle count_pkt (int dst, int proto) {
    int idx = Array.get(nexthops, dst);
    if (proto != TCP) {
        if (proto == UDP)
            idx = idx + NUM_PORTS;
        else
            idx = idx + NUM_PORTS_X2;
    }
    Array.set(pcts, idx, plus, 1);
    if (proto == TCP)
        Array.set(hcts, dst, plus, 1);
}
```

Figura 34 Ejemplo Lucid

2.3.3 GRAPH TO P4

Otro ejemplo de programación a alto nivel para P4 es, por ejemplo, el trabajo de Eder Ollora Zaballa y Zifan Zhou, que en 2019 presentaban un generador de grafos a P4. Su propuesta presenta un toolset que permite un nivel de abstracción más al lenguaje de programación de alto nivel (el mismo objetivo que tiene este trabajo) [17].

El motivo que argumentan los creadores de esta herramienta es la elevada curva de dificultad que tiene P4 para la gente que lo ve por primera vez. Pueden usar está herramienta para comenzar el código de manera más sencilla y luego desarrollar el código específico del plano de datos como las acciones de manera “manual”.

3. ANÁLISIS Y DISEÑO

Tras haber analizado las herramientas que utilizaremos, así como el contexto de los desarrollos e investigaciones que se están realizando actualmente en las redes definidas por software, en este apartado expondremos qué vamos a hacer, cómo lo vamos a hacer y por qué lo vamos a hacer.

En el anterior apartado hemos resumido brevemente el trabajo de Victoria en el cual nos hemos basado para hacer este proyecto, no obstante, aún debemos abordar las diferencias.

Victoria elige Ryu, el cual está desarrollado en Python, lo cual ya supone una ventaja a la hora de trabajar también con Blockly que usa Python, es fácil de usar y muy sencillo desarrollar con él, para el objetivo que se propone Victoria es suficiente. Por otro lado, es la principal limitación ya que a la hora de acercarse a un despliegue real de una red de telecomunicación será necesario utilizar otro framework más real como ONOS o OpenDaylight que son más complejos de utilizar y de desarrollar con ellos pero por otro lado tienen numerosas ventajas como Command-line Interface, interfaz gráfica (ya hemos visto como ONOS tiene una interfaz gráfica muy completa en el apartado de estado del arte), poder activar o desactivar aplicaciones en tiempo de ejecución…

Es por todos estos motivos que en nuestro trabajo hemos optado por un controlador comercial más potente, ahí es cuando nos dividimos entre ONOS y OpenDaylight, los dos controladores Open Source más destacados.

Por recomendación de la tutora optamos por ONOS debido a que la curva de aprendizaje es algo más sencilla, ya que la arquitectura de ODL es muy generalizada y compleja para implementar nuevos módulos.

En el apartado de la interfaz visual mantenemos el uso de Blockly viendo que encaja perfectamente con lo que buscamos: crear una interfaz visual que sea sencilla de usar y poder asociar un código para crear los mensajes OpenFlow.

El hecho de haber elegido modificar el framework de trabajo utilizado hace que el código desarrollado por Victoria no nos sirva, no obstante, sí que hay una parte de Blockly que sí que podemos reutilizar, los propios bloques, Victoria crea cuatro bloques, cada uno representa un mensaje OpenFlow: PacketIn, PacketOut, PortStatus y FlowMod.

En nuestro caso solo utilizaremos dos de ellos, PacketIn y PacketOut, modificaremos una demo de Blockly al igual que hizo Victoria para incluir estos bloques y crearemos una nueva pestaña llamada Java para Onos, tendremos que desarrollar un código que haga que los paquetes generen en esta nueva pestaña un controlador compilable en Java para ONOS con los mensajes OpenFlow y los inputs elegidos en Blockly.

Una vez tengamos el controlador tendremos que exportarlo a una aplicación ONOS, en este caso partimos de la aplicación de OnePing la cual hemos modificado bastante, originalmente esta aplicación controlaba los pings realizados en una red y si llegaban más de 1 igual, los paraba y avisaba por consola. Hemos modificado esta aplicación para crear un controlador que maneja los paquetes a nuestro gusto en función de los mensajes OpenFlow que queramos.

4. IMPLEMENTACIÓN Y DESARROLLO

Comenzamos este apartado de desarrollo con un pequeño recordatorio del objetivo del trabajo y el planteamiento para este.

El objetivo es utilizar Mininet para desplegar una red conectada a nuestro framework ONOS, este framework nos permite desarrollar aplicaciones en Java sobre la red desplegada, por lo que, programaremos una app que maneje el tráfico de la red con los mensajes PacketIn y PacketOut, una vez terminada la aplicación, descargaremos Blockly, creando un workspace modificando una de las demos disponibles y añadiremos el código de dicha aplicación a los bloques de Victoria, creando una interfaz en Blockly que nos permitirá programar de una manera muy sencilla redes SDN.

4.1 INSTALACIÓN DE LAS HERRAMIENTAS

4.1.1 Mininet

La instalación de Mininet es muy sencilla, tan solo tenemos que instalar Git, clonar el GitHub oficial de Mininet, elegir versión y ejecutar el install.sh, en este caso con la opción -a para instalar todos los complementos [18].

```
alvaro@tfg:~$ sudo apt install git
[sudo] password for alvaro:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk
  gitweb git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 4.720 kB of archives.
After this operation, 33,8 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://es.archive.ubuntu.com/ubuntu bionic/main amd64 liberror-perl all 0.17025-1 [22,8 kB]
Get:2 http://es.archive.ubuntu.com/ubuntu bionic/main amd64 git-man all 1:2.17.0-1ubuntu1 [803 kB]
Get:3 http://es.archive.ubuntu.com/ubuntu bionic/main amd64 git amd64 1:2.17.0-1ubuntu1 [3.894 kB]
Fetched 4.720 kB in 1s (4.022 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 164576 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17025-1_all.deb ...
Unpacking liberror-perl (0.17025-1) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1%3a2.17.0-1ubuntu1_all.deb ...
Unpacking git-man (1:2.17.0-1ubuntu1) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1%3a2.17.0-1ubuntu1_amd64.deb ...
Unpacking git (1:2.17.0-1ubuntu1) ...
Setting up git-man (1:2.17.0-1ubuntu1) ...
Setting up liberror-perl (0.17025-1) ...
Setting up git (1:2.17.0-1ubuntu1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
alvaro@tfg:~$ git clone git://github.com/mininet/mininet
Cloning into 'mininet'...
remote: Enumerating objects: 10165, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 10165 (delta 2), reused 7 (delta 2), pack-reused 10154
Receiving objects: 100% (10165/10165), 3.19 MiB | 5.76 MiB/s, done.
Resolving deltas: 100% (6784/6784), done.
alvaro@tfg:~$ cd mininet
alvaro@tfg:~/mininet$ git checkout -b mininet-2.3.0 2.3.0
Switched to a new branch 'mininet-2.3.0'
alvaro@tfg:~/mininet$ cd ..
alvaro@tfg:~$ mininet/util/install.sh -a
Detected Linux distribution: Ubuntu 18.04 bionic amd64
```

Figura 35 Instalación de Mininet

Una vez instalado para iniciarlo tan sólo hay que escribir el comando “sudo mn” y las opciones que queramos:

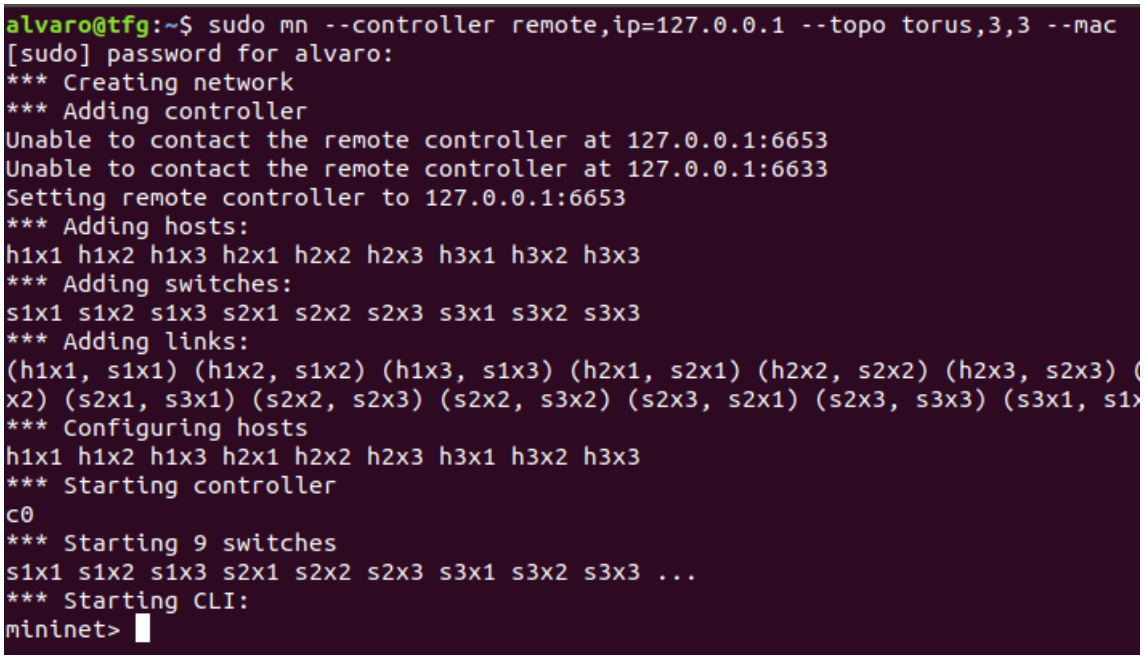


Figura 36 Lanzamiento de Mininet

En este caso elegimos el controlador remoto, la IP y la topología que queremos, una vez iniciado Mininet nos deja una consola donde podemos controlar la red, con esta consola podremos utilizar todos los comandos mencionados en el apartado de estado del arte, como hacer pings, activar o desactivar enlaces, lanzar un Xterm…

4.1.2 ONOS

La instalación de ONOS es bastante más compleja que Mininet o el resto de las herramientas, para empezar, es importante mencionar que hemos seguido la siguiente guía rápida de instalación, ya que en internet e incluso en la propia página oficial de ONOS hay bastantes guías y todas con distintos métodos de instalación dependiendo de las necesidades de cada usuario, en nuestro caso hemos seguido el inicio rápido para desarrolladores [19].

Dentro de esta guía se nos menciona en un pequeño apartado que es necesario instalar Bazelisk y una serie de herramientas y requisitos antes de comenzar la instalación de ONOS [20].

En estos requisitos se pide que el sistema tenga 16GB de RAM para poder utilizar ONOS con Bazel como hemos hecho nosotros, no obstante, en ninguno de los dos ordenadores donde hemos estado haciendo el desarrollo de este trabajo teníamos esta cantidad (4 en uno y 6 en otro) y todo ha funcionado correctamente, por lo que no parece un requisito muy necesario, por otro lado, se nos pide Linux o MacOS, en nuestro caso estamos utilizando Ubuntu 18.04.

Tras esto ya pasamos a la parte del software necesario, comenzamos instalando Python2 y Python 3, tal y como se nos pide.

Install Python

```
sudo apt install python
sudo apt install python3
sudo apt install python-pip python-dev python-setuptools
sudo apt install python3-pip python3-dev python3-setuptools

pip3 install --upgrade pip
pip3 install selenium
```

Figura 37 Instalación Python

Tras la ejecución de estos comandos debemos instalar Git y git-review, en este caso git ya se encontraba instalado tras la instalación de Mininet.

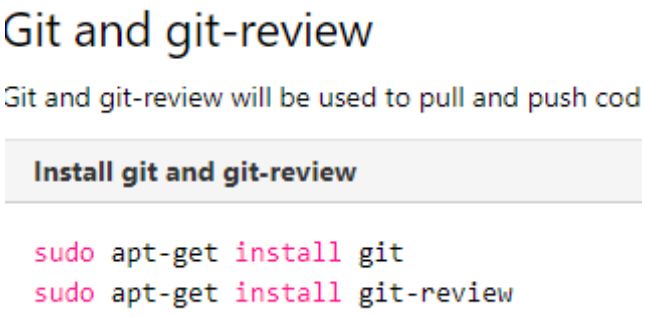


Figura 38 Instalación Git y Git-review

Una vez instaladas todas estas herramientas pasamos a la instalación de Bazelisk, este es un wrapper para Bazel, programa que necesitamos para instalar y lanzar ONOS.

Se instala cogiendo el directorio de GitHub y dando permisos y moviéndolo a la carpeta indicada en /usr/local/bin/bazel.

Tras esto ejecutamos el comando bazel versión que servirá para instalar la herramienta y que nos diga en que versión se ha instalado, en nuestro caso en la 1.4.0.

```
alvaro@tfg:~/onos$ wget https://github.com/bazelbuild/bazelisk/releases/download/v1.4.0/bazelisk-linux-amd64
```

Figura 39 Instalación Bazelisk 1

```
alvaro@tfg:~/onos$ chmod +x bazelisk-linux-amd64
alvaro@tfg:~/onos$ sudo mv bazelisk-linux-amd64 /usr/local/bin/bazel
```

Figura 40 Instalación Bazelisk 2

```
alvaro@tfg:~/onos$ bazel version
Bazelisk version: v1.4.0
Extracting Bazel installation...
Starting local Bazel server and connecting to it...
Build label: 3.7.2
Build target: bazel-out/k8-opt/bin/src/main/java/com/google/devtools/build/lib/bazel/BazelServer_deploy.jar
Build time: Thu Dec 17 16:57:23 2020 (1608224243)
Build timestamp: 1608224243
Build timestamp as int: 1608224243
```

Figura 41 Instalación Bazelisk 3

Una vez instalado Bazelisk pasamos a la instalación de ONOS, de nuevo se nos pide otra serie de herramientas antes de comenzar:

Other dependencies

```
git
zip
curl
unzip
python # 2.7 required by some development scripts
python3 # Required by Bazel
bzip2 # Needed by legacy GUI build
```

Figura 42 Dependencias de ONOS

En nuestro caso ya tenemos instalado la mayoría de estas dependencias, procedemos a instalar aquellas que nos faltan, todas aquellas que nos falten se instalan con el comando sudo apt-get install “herramienta” y en caso de ya estar instalada y ejecutar el comando lo único que ocurre es que se actualiza la herramienta a la última versión:

```
alvaro@tfg:~$ sudo apt-get install curl
```

Figura 43 Instalación Curl

Una vez finalizado todo este proceso podemos pasar a la instalación de ONOS, comenzamos clonando el GitHub:

```
alvaro@tfg:~$ git clone https://gerrit.onosproject.org/onos
```

Figura 44 Clonación GitHub ONOS

Una vez clonado se nos crea una carpeta llamada ONOS con todos los archivos, nos desplazamos a esta usando el comando “cd onos” y ejecutamos lo siguiente para que comience la instalación:

```
alvaro@tfg:~/onos$ bazel build onos
```

Figura 45 Instalación ONOS

La ejecución de este comando tarda bastante tiempo, unos minutos en el caso de nuestro hardware que comentaremos al final de la memoria, con esto finalizamos la instalación de las dos herramientas y podemos pasar al desarrollo del proyecto.

4.2 MININET

Comenzamos el desarrollo del proyecto eligiendo la topología de la red que vamos a utilizar, realmente podemos utilizar cualquier esquema ya que lo único que va a hacer nuestra aplicación son mensajes PacketIn y PacketOut, no obstante, para hacer todas las pruebas correspondientes vamos a utilizar una topología sencilla pero que tenga varios elementos.

Ya hemos hablado en el apartado del estado del arte que en Mininet podemos desarrollar un pequeño archivo Python donde añadimos los switches, hosts y enlaces y la red simulada tendrá esa topología, en este caso hemos utilizado una topología triangular que se encuentra en el GitHub de la tutora de este TFG, Elisa Rojas [21].

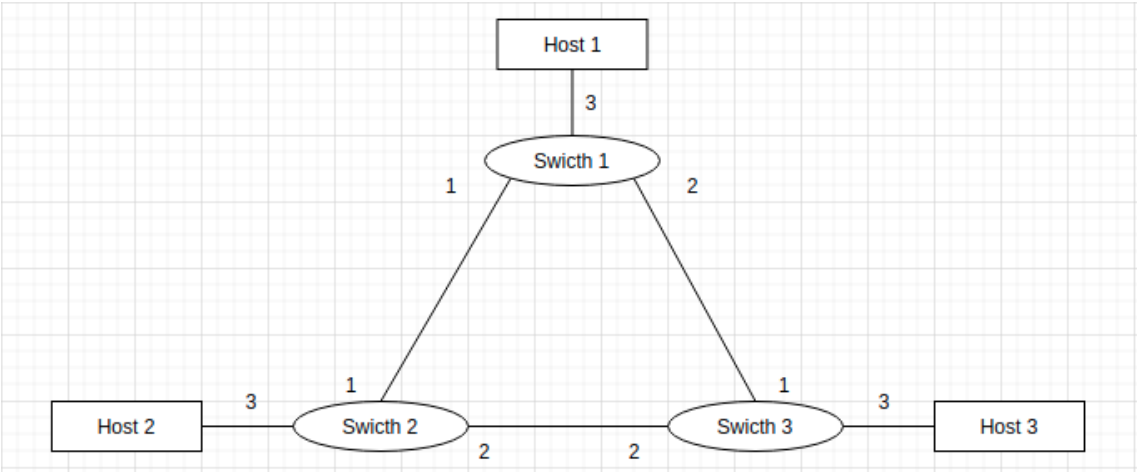


Figura 46 Topología del trabajo

Es importante destacar los puertos, cada switch está conectado con su host con el puerto 3, mientras que con los otros dos switches correspondientes con los puertos 1 y 2, de esta manera, si un paquete viaja, por ejemplo, del host 1 al host 2 pasaría por los siguientes puertos:

1. Puerto 3 del Switch 1.
2. Puerto 1 del Switch 1.
3. Puerto 1 del Switch 2.
4. Puerto 3 del Switch 2.

Procedemos a la comprobación:

En primer lugar, lanzamos Mininet con el siguiente comando:

```
alvaro@tfg:~$ sudo mn --controller remote,ip=127.0.0.1 --custom ~/mininet/custom/triangle.py --topo triangle --mac
```

Figura 47 Lanzamiento de Mininet

indicando la ruta donde se encuentra la topología personalizada, en este caso en /Mininet/custon/triangle.py.

Posteriormente iniciamos ONOS, nos movemos a la carpeta donde se encuentra y lanzamos con bazel el comando run onos-local con las opciones clean y debug.

```
alvaro@tfg:~$ cd onos
alvaro@tfg:~/onos$ bazel run onos-local -- clean debug
```

Figura 48 Lanzamiento ONOS

Y nos conectamos a ONOS para poder activar las aplicaciones de OpenFlow y Forwarding.

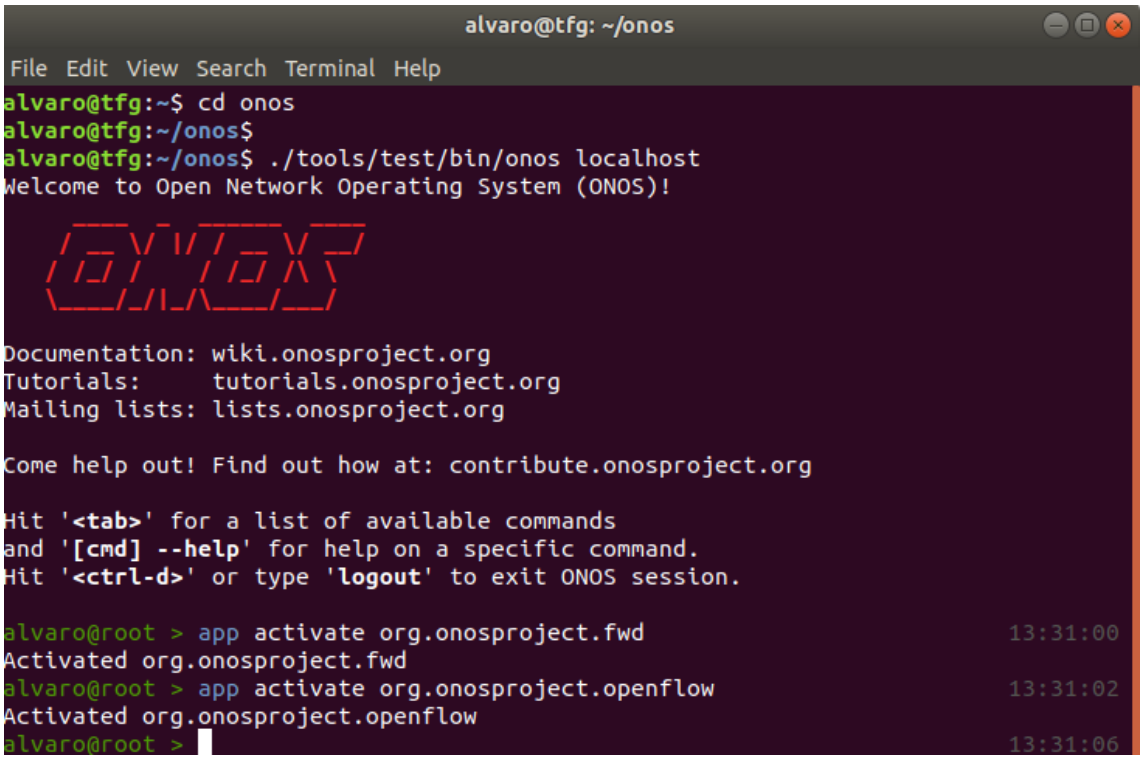


Figura 49 Activación de openflow y fwd en ONOS

Ahora podemos volver a la consola de Mininet.

Con el comando xterm que mencionamos en el apartado de estado del arte cuando hablamos de Mininet, dijimos que podíamos abrir una miniconsola con la que ver un dispositivo concreto, en este caso abriremos una para ver lo que ocurre en el switch 1.

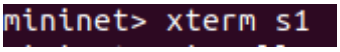


Figura 50 Lanzamiento Xterm Mininet

Figura 51 Xterm S1

En esta consola, podemos utilizar el comando “ifconfig” para que nos muestro los puertos disponibles.

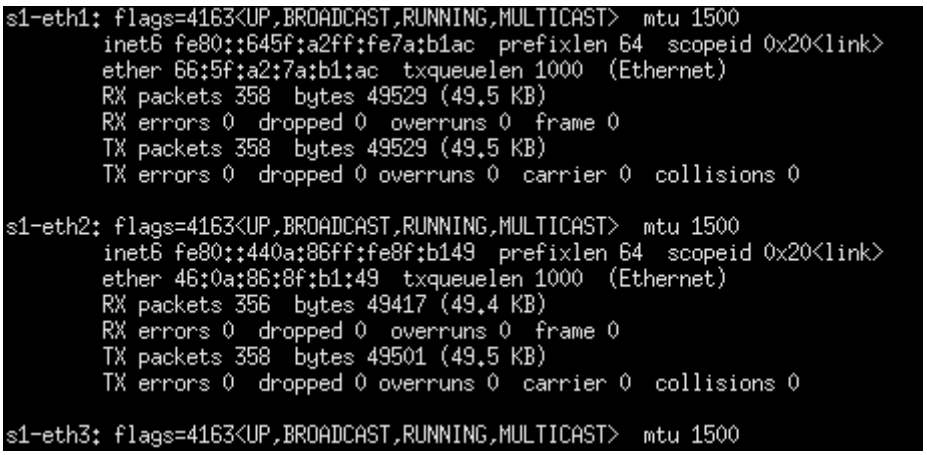


Figura 52 Comando ifconfig

Podemos ver como todos estos puertos se refieren al switch 1 (de ahí el inicio s1) y eth1 se refiere al puerto 1, eth2 al puerto 2 y eth3 al puerto 3.

De esta manera si queremos captar el tráfico de uno de ellos debemos usar el comando “tcpdump” indicando el switch y el puerto.

Figura 53 Comando tcpdump

De regreso a la consola de Mininet, mandamos un paquete de h1 a h2.

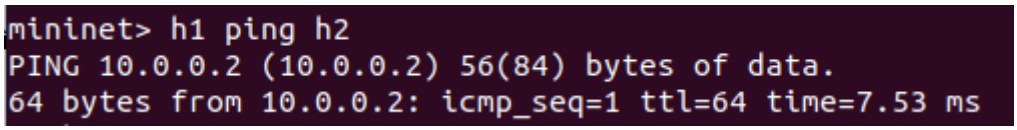


Figura 54 Ping en Mininet

Volviendo a la consola que hemos lanzado con el comando xterm en el switch 1 y con el comando "tcpdump -i s1-eth3" para ver el puerto 3 del switch 1, podemos ver como el ping pasa por aquí.

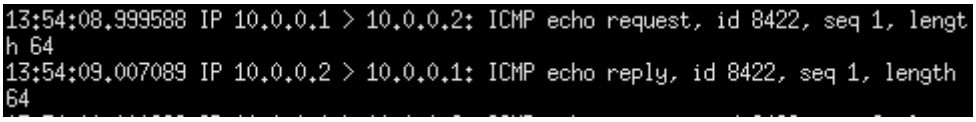


Figura 55 Consola del switch 1

Si repetimos la acción con todos los puertos mencionados podríamos ver como pasa el mismo paquete por ello, pero también vamos a comprobar que el paquete no pasa por los puertos que no tiene que pasar, por ejemplo, el puerto 2 del switch 1.

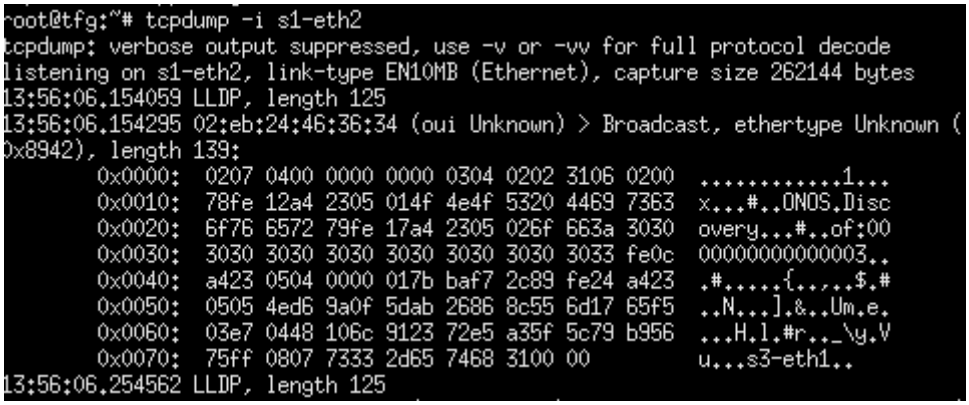


Figura 56 Consola del switch 1 puerto 2

Como podemos observar el paquete no pasa por este puerto, tan solo aparece un paquete Discovery de ONOS que aparece cada 3 segundos.

4.3 ONOS

Entendiendo la topología pasemos a la parte de ONOS y sus aplicaciones.

ONOS cuenta con un pequeño tutorial de como desplegar aplicaciones en su framework, el cual hemos seguido para comprender el desarrollo de aplicaciones en ONOS [22].

Con este tutorial creamos una aplicación vacía y vemos cómo funciona el compilado y la instalación de aplicaciones en ONOS, no obstante, desarrollar de 0 una aplicación en ONOS no es nada sencillo, es por esto por lo que hemos decidido buscar una aplicación que tuviera una estructura similar y modificarla hasta conseguir la que queremos.

El GitHub oficial de ONOS tiene varias aplicaciones de ejemplo [23].

Concretamente, dentro del conjunto de aplicaciones disponibles, se ha elegido la aplicación de OnePing. Está aplicación tiene un funcionamiento muy sencillo donde sólo permite un ping por minuto por cada par de MACs, es decir en caso de que haya un ping entre dos dispositivos todos los pings que lleguen detrás en un intervalo de un minuto se descartan.

Para entender el código, lo cual es importante para saber lo que vamos a modificar y por qué hay que entender tres conceptos: Packet Processor, Traffic Selector y Traffic Treatment.

Packet Processor: Se encarga de decidir qué hacer cuando llega un paquete a un dispositivo SDN.

Traffic Selector: actúa como un match, añadimos una o varias condiciones, por ejemplo, si vienen de un cierto destino o van a una determinada dirección, si los paquetes procesados cumplen estas condiciones realizan una cierta acción.

Traffic Treatment: los paquetes que cumplen las condiciones del selector pasan al treatment donde se indica la acción a realizar, por ejemplo, mandar el paquete por un determinado puerto o descartarlo.

En nuestra aplicación el Packet Processor descartará todos los paquetes que no sean ICMP, esto no ha sido modificado de la aplicación de OnePing original,

El traffic selector en la aplicación original funcionaba comparando la MAC dst y la MAC src a partir del segundo ping, en caso de que se cumpliera el treatment realizaba la acción drop() ignorando el paquete.

Nosotros hemos modificado estos dos últimos de manera que el traffic selector funcione de PacketIn utilizando las condiciones establecidas en Blockly (MAC dst y el switch) y en caso de que cumpla la condición el TrafficTreatment realizará la acción de PacketOut escogiendo el puerto por el que mandar el paquete.

El código desarrollado lo mostraremos más adelante, ya que una parte de este código está dentro del código de los bloques de Blockly y otro en el archivo code.js de Blockly.

4.4 BLOCKLY

Blockly es la interfaz visual de nuestro proyecto, la idea es que lo único que tenga que hacer el usuario es mover unas piezas de puzle en una interfaz web y elegir en ellas unos pocos parámetros, con esa sencillez se debería crear un controlador de redes definidas por software que ejecute tales acciones.

Comenzamos la parte de Blockly descargándolo, se encuentra en el GitHub oficial [24].

No es necesario realizar ninguna instalación, tan sólo descomprimir y nos encontraremos las siguientes carpetas:

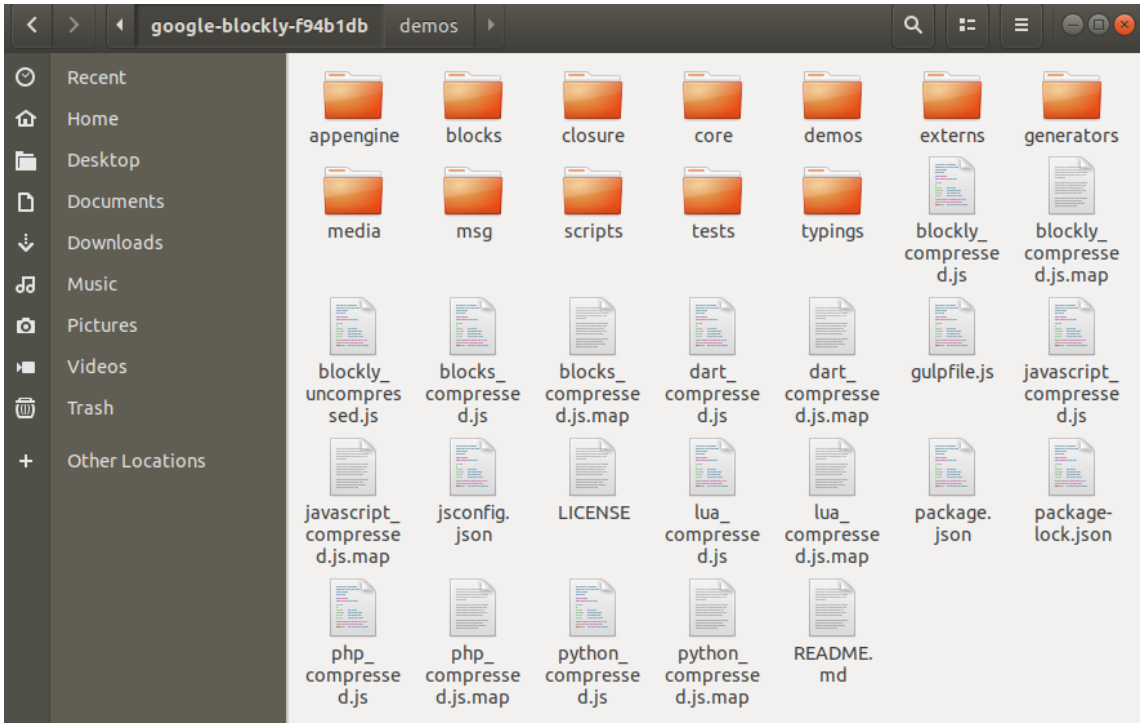


Figura 57 Ficheros Blockly

Así pues, lo primero que debemos de hacer es crear nuestros bloques personalizados con la herramienta de Blockly Developer Tools, en el apartado de Blockly Factory podemos crearlos [25].

En nuestro caso debemos de crear dos bloques: PacketIn y PacketOut, no obstante, como hemos comentado en apartados anteriores, estos bloques ya los ha creado Victoria y podemos utilizar los suyos.

4.4.1 PacketIn

El PacketIn tiene la siguiente estructura:

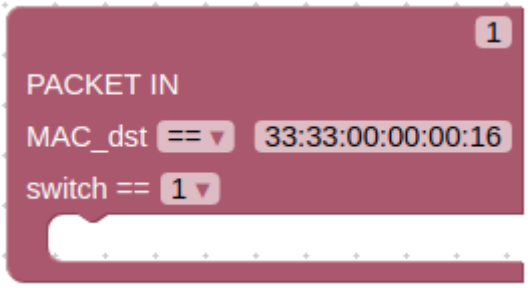


Figura 58 Bloque PacketIn

Como podemos ver tiene una opción para elegir si queremos comparar la coincidencia o no coincidencia de la MAC destino, hay que recordar que el PacketIn será traducido en el controlador como un TrafficSelector, en caso de que cumpla esta condición se efectuará la acción de la pieza que tenga dentro (PacketOut), la otra variable con la que cuenta este mensaje es la del switch en el que se efectuará el PacketIn.

Y es creado en de la siguiente manera:

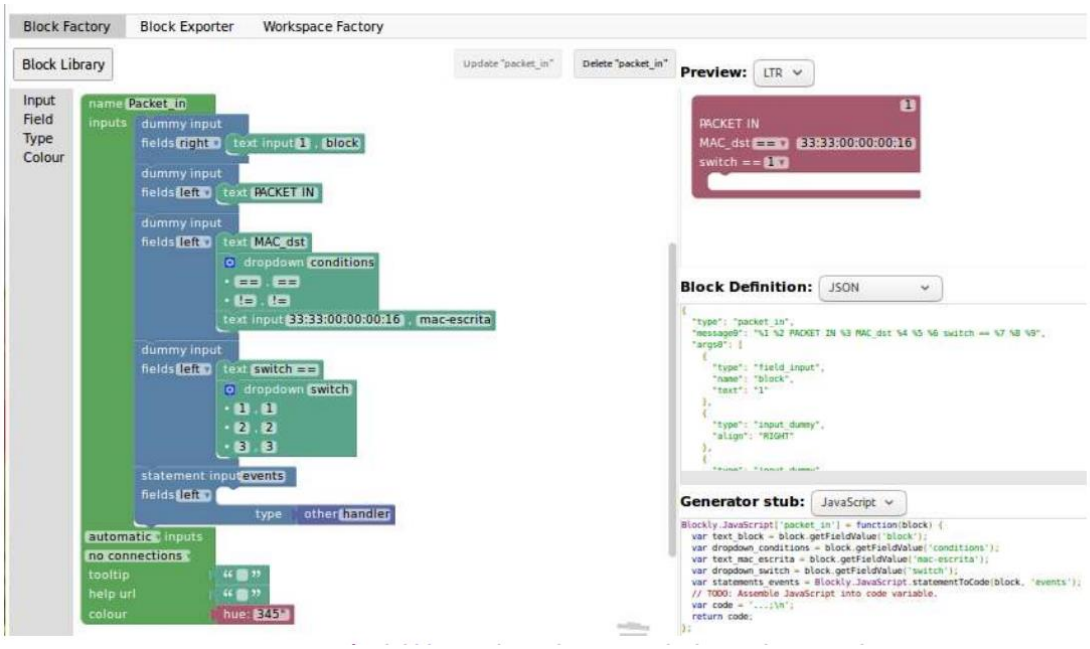


Figura 59 Creación del bloque PacketIn

Lo cual genera el siguiente código (estructura del bloque):

```
Blockly.Blocks['packet_in'] = {
```

```
  init: function() {
    this.appendDummyInput()
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldTextInput("1"), "block");
    this.appendDummyInput()
        .appendField("PACKET IN");
    this.appendDummyInput()
        .appendField("MAC_dst")
        .appendField(new Blockly.FieldDropdown([["==","=="],
["!=","!="]]), "conditions")
        .appendField(new
Blockly.FieldTextInput("33:33:00:00:00:16"), "mac-escrita");
    this.appendDummyInput()
        .appendField("switch ==")
        .appendField(new Blockly.FieldDropdown([["1","1"],
["2","2"], ["3","3"]]), "switch");
    this.appendStatementInput("events")
        .setCheck("handler");
    this.setColour(345);
 this.setTooltip("");
 this.setHelpUrl("");
  }
};
```

El código generado por Blockly obviamente no nos sirve, debemos modificarlo para que genere el trozo de código para ONOS correspondiente cogiendo las variables del bloque, este código está definido dentro de la carpeta de “Generators” donde se encuentran los generadores de código de cada bloque el código es el siguiente:

```
Blockly.Python['packet_in'] = function(block) {
  var text_block = block.getFieldValue('block');
  var dropdown_conditions = block.getFieldValue('conditions');
  var text_mac_escrita = block.getFieldValue('mac-escrita');
  var dropdown_switch = block.getFieldValue('switch');
  var statements_events = Blockly.Python.statementToCode(block,
'events');

Blockly.Python['packet_in'] = function(block) {
  var text_block = block.getFieldValue('block');
  var dropdown_conditions = block.getFieldValue('conditions');
  var text_mac_escrita = block.getFieldValue('mac-escrita');
  var dropdown_switch = block.getFieldValue('switch');
  var statements_events = Blockly.Python.statementToCode(block,
'events');

  var code = '  \n# PACKET IN\n'+
             '    private void packetin(DeviceId deviceId,
MacAddress src, MacAddress dst, PacketContext context) {\n' +
             '        TrafficSelector selector =
DefaultTrafficSelector.builder()\n'+
```

```
                '
.matchEthSrc(src).matchEthDst(dst).build();\n'+
                '        if (deviceId.toString().equals(\"' +
'of:0000000000000000' + dropdown_switch + '\")) {\n'+
                '            if (dst.toString().equals(\"' +
text_mac_escrita + '\")) {\n'+
                '        ' +statements_events ;
  return code;
};
```

4.4.2 PacketOut

El PacketOut tiene la siguiente estructura:

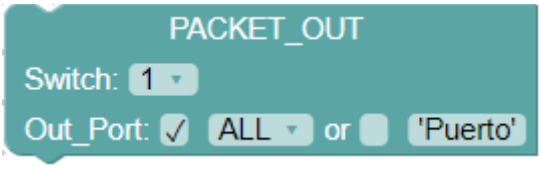


Figura 60 Bloque PacketOut

De nuevo en el PacketOut podemos elegir el Switch sobre el que realizará el PacketOut y por otro lado podemos elegir entre mandarlo por todos los puertos o eligiendo un puerto concreto, en nuestro trabajo siempre elegiremos esta última opción, seleccionando con el tick la parte derecha del OR y escribiendo a mano un puerto de salida, el bloque es heredado del trabajo de Victoria, pero en nuestro caso no hemos implementado la opción de lanzarlo por todos los puertos.

Y es creado de la siguiente manera:

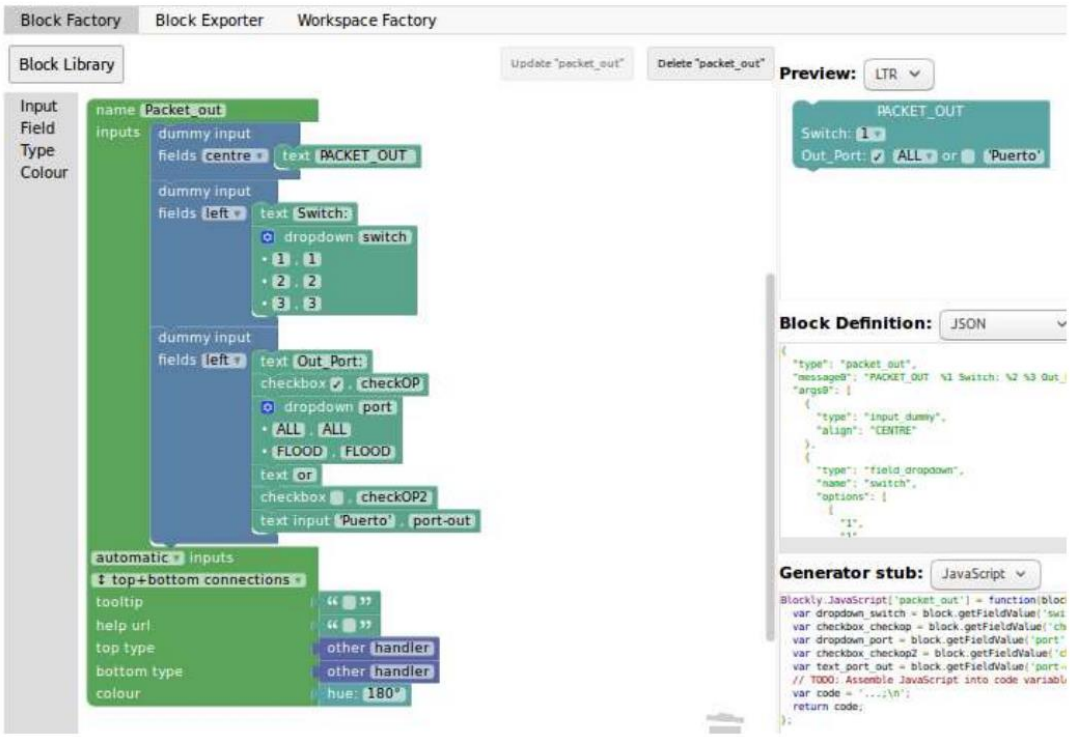


Figura 61 Creación del bloque PacketOut

El Código de la estructura del bloque es el siguiente:

```
Blockly.Blocks['packet_out'] = {
  init: function() {
    this.appendDummyInput()
        .setAlign(Blockly.ALIGN_CENTRE)
        .appendField("PACKET_OUT ");
    this.appendDummyInput()
        .appendField("Switch:")
        .appendField(new Blockly.FieldDropdown([["1","1"],
["2","2"], ["3","3"]]), "switch");
    this.appendDummyInput()
        .appendField("Out_Port:")
        .appendField(new Blockly.FieldCheckbox("TRUE"),
"checkOP")
        .appendField(new Blockly.FieldDropdown([["ALL","ALL"],
["FLOOD","FLOOD"]]), "port")
        .appendField("or")
        .appendField(new Blockly.FieldCheckbox("FALSE"),
"checkOP2")
        .appendField(new Blockly.FieldTextInput("'Puerto'"),
"port-out");
    this.setPreviousStatement(true, "handler");
    this.setNextStatement(true, "handler");
    this.setColour(180);
 this.setTooltip("");
 this.setHelpUrl("");
  }
};
```

Y el código que genera:

```
Blockly.Python['packet_out'] = function(block) {
  var dropdown_switch = block.getFieldValue('switch');
  var checkbox_checkop = block.getFieldValue('checkOP') ==
'TRUE';
  var dropdown_port = block.getFieldValue('port');
  var checkbox_checkop2 = block.getFieldValue('checkOP2') ==
'TRUE';
  var text_port_out = block.getFieldValue('port-out');

  // TODO: Assemble Python into code variable.
  // FALTA AÑADIR COSAS DEL SWITCH
  var code = '
context.treatmentBuilder().setOutput(PortNumber.portNumber(2));\
n'+
              '                context.send();\n' +
              '             }\n'+
              '         }\n'+
             '         TrafficTreatment drop =
DefaultTrafficTreatment.builder()\n'+
              '
.setOutput(PortNumber.portNumber(2)).build();\n'+
              '         flowObjectiveService.forward(deviceId,
DefaultForwardingObjective.builder()\n'+
              '                .fromApp(appId)\n'+
              '                .withSelector(selector)\n'+
              '                .withTreatment(drop)\n'+
              '
.withFlag(ForwardingObjective.Flag.VERSATILE)\n'+
              '                .withPriority(DROP_PRIORITY)\n'+
              '                .makeTemporary(TIMEOUT_SEC)\n'+
              '                .add());\n'+
              '    }\n'+
              '}\n';
  return code;
};
```

El código de la estructura de los bloques va definido en la carpeta “Blocks” en el archivo “Handlers”.

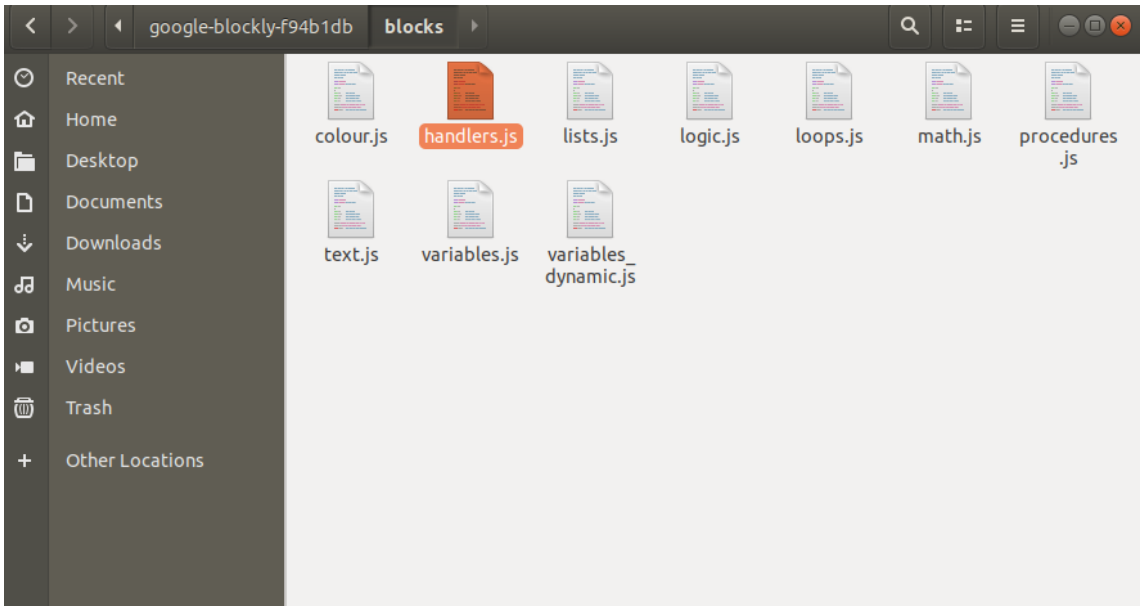


Figura 62 Archivos Blockly carpeta blocks

Por otro lado, el código que genera que hemos desarrollado nosotros va incluido en la carpeta “Generators” aquí es donde se traduce Blockly el bloque a código.

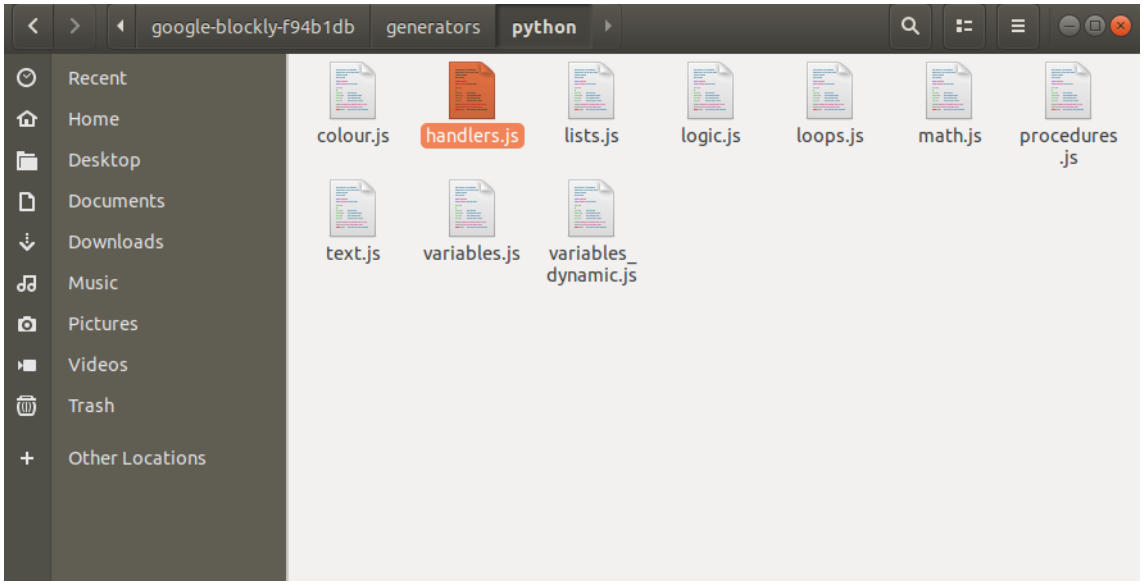


Figura 63 Archivos Blockly generators/python

Una vez tenemos terminados los bloques vamos a preparar el workspace, partimos de la demo de CODE, esta demo funciona de manera que tenemos en el panel de la

izquierda una serie de bloques que imitan a la lógica de los lenguajes de programación (if, else, count…) en la parte central arrastramos estos bloques y los conectamos de manera que en la parte de la izquierda se genera un código del que podemos elegir el lenguaje (exactamente lo que queremos nosotros).

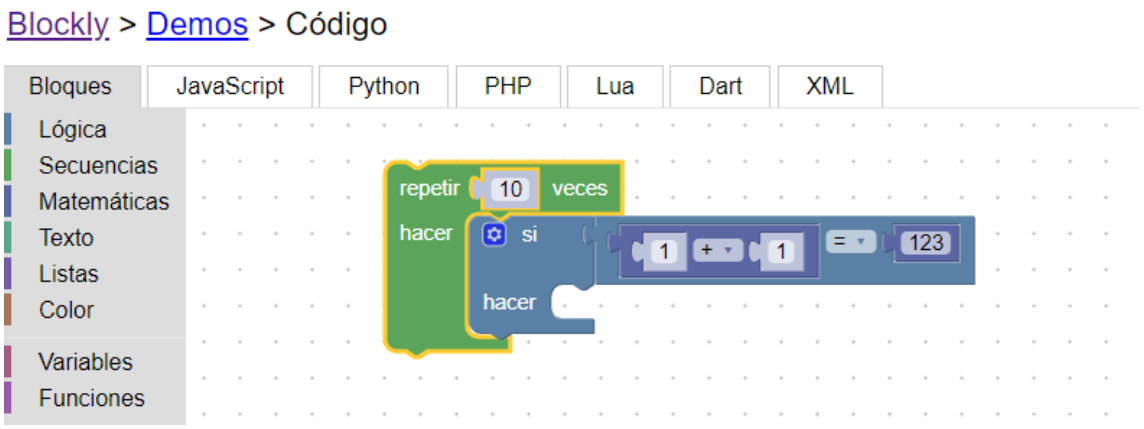


Figura 64 Demo Code Blockly

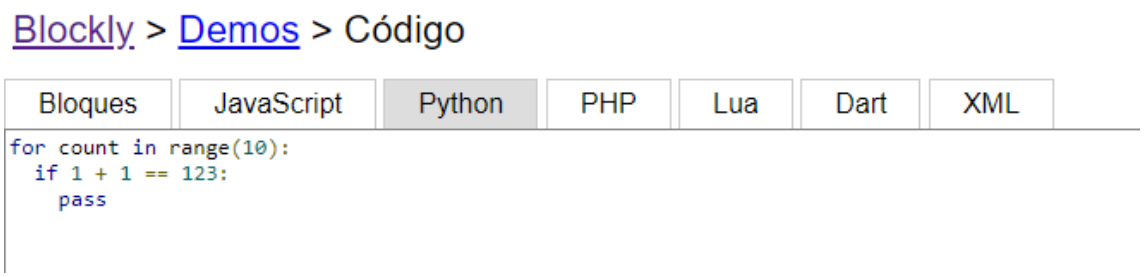


Figura 65 Demo Code Blockly 2

La estructura de la aplicación es la siguiente:

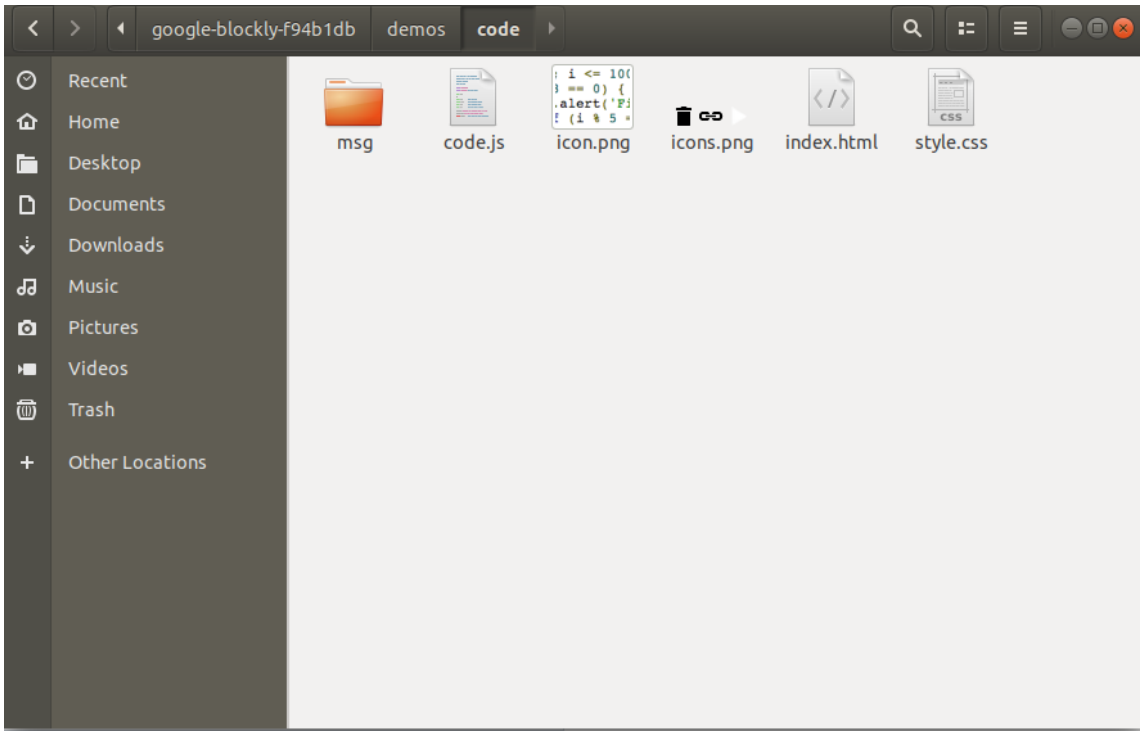


Figura 66 Archivos de la demo CODE

Debemos modificar index.html para incluir dos apartados nuevos, uno llamado Handlers donde se mostrarán los bloques personalizados que creó Victoria y otro llamado Java para ONOS donde tras usar estos bloques aparecerá el código del controlador en ONOS que hemos creado, este código será dinámico y se adaptará a los bloques que pongamos con los parámetros que elijamos.

En esta primera modificación estamos incluyendo los bloques que hemos introducido en el archivo handlers.js en la carpeta de bloques, aquí está definido como son los bloques, que variables tienen, que forma, que color… y a su vez estamos incluyendo el archivo que se encuentra en la carpeta de generators, donde hemos incluido el código que tienen que crear estos bloques al usarlos en nuestra interfaz visual.

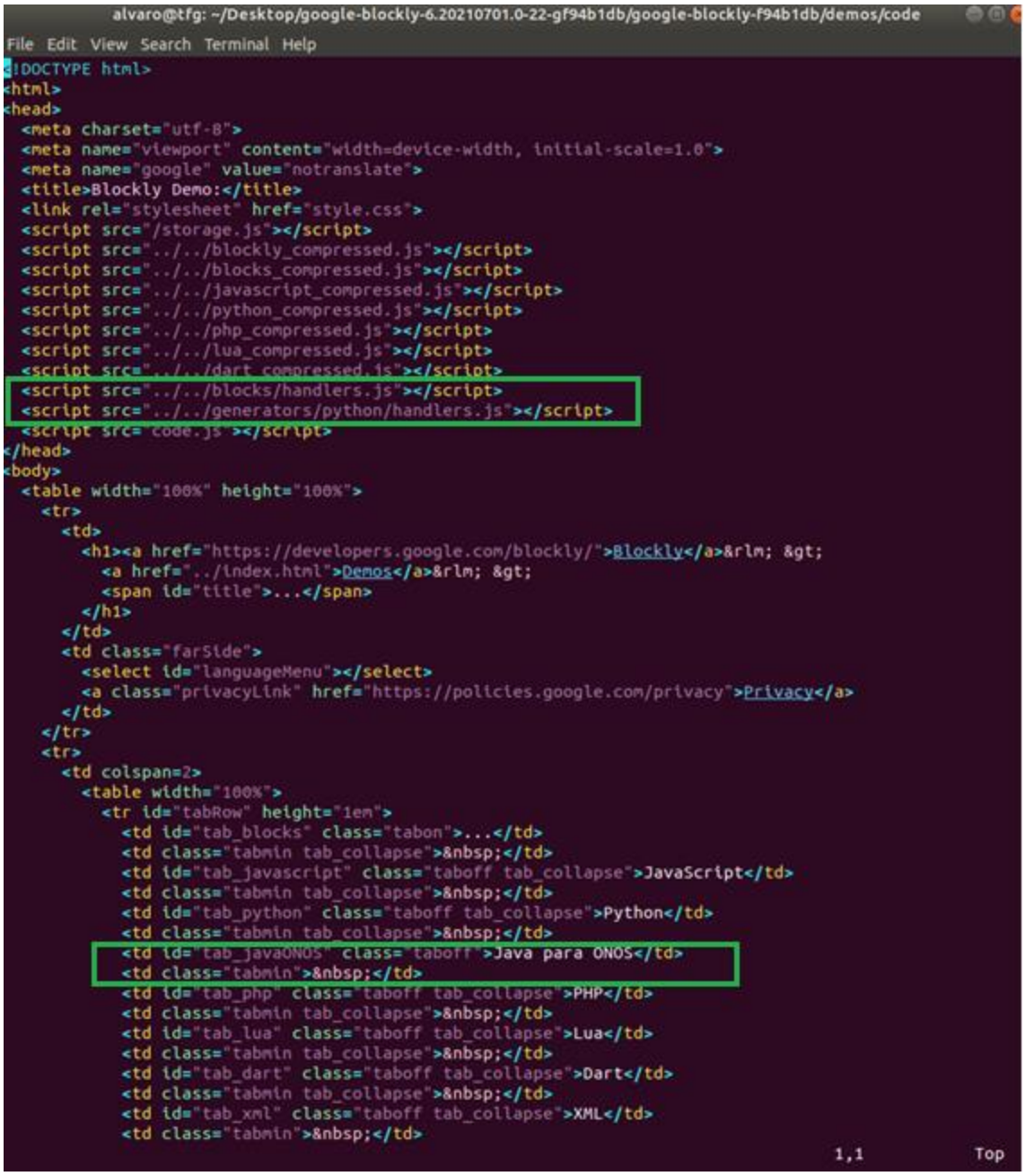


Figura 67 Modificación index.html 1

En esta segunda modificación estamos incluyendo la pestaña de java para ONOS:

```
<div id="content_blocks" class="content"></div>
<pre id="content_javascript" class="content prettyprint lang-js"></pre>
<pre id="content_python" class="content prettyprint lang-py"></pre>
<pre id="content_javaONOS" class="content"></pre>
<pre id="content_php" class="content prettyprint lang-php"></pre>
<pre id="content_lua" class="content prettyprint lang-lua"></pre>
<pre id="content_dart" class="content prettyprint lang-dart"></pre>
<textarea id="content_xml" class="content" wrap="off"></textarea>
```

Figura 68 Modificación index.html 2

Y en esta ultimo estamos incluyendo los dos bloques creados (PacketIn y PacketOut) en la nueva pestaña de handlers:

```
    <category name="%{BKY_CATVARIABLES}" colour="%{BKY_VARIABLES_HUE}" custom="VARIABLE"></category>
    <category name="%{BKY_CATFUNCTIONS}" colour="%{BKY_PROCEDURES_HUE}" custom="PROCEDURE"></category
>
    <category name="Handlers" colour="#a5a55b">
    <block type="packet_in"></block>
    <block type="packet_out"></block>
    </category>
  </xml>

</body>
</html>
                                                                              403,7         Bot
```

Figura 69 Modificación index.html 3

También debemos modificar el código de code.js, aquí es donde incluiremos el resto del código del controlador, comenzamos incluyendo la pestaña de java para ONOS:

```
Code.TABS_ = ['blocks', 'javascript', 'php', 'python', 'javaONOS', 'dart', 'lua', 'xml'];
```

Figura 70 Modificación code.js 1

Y eligiendo el generador de código Python para esta nueva pestaña, recordamos que los generadores de código los hemos metido dentro de la carpeta generators/python:

```
Code.renderContent = function() {
  var content = document.getElementById('content_' + Code.selected);
  // Initialize the pane.
  if (content.id == 'content_xml') {
    var xmlTextarea = document.getElementById('content_xml');
    var xmlDom = Blockly.Xml.workspaceToDom(Code.workspace);
    var xmlText = Blockly.Xml.domToPrettyText(xmlDom);
    xmlTextarea.value = xmlText;
    xmlTextarea.focus();
  } else if (content.id == 'content_javascript') {
    Code.attemptCodeGeneration(Blockly.JavaScript, 'js');
  } else if (content.id == 'content_python') {
    Code.attemptCodeGeneration(Blockly.Python, 'py');
  } else if (content.id == 'content_javaONOS') {
    Code.attemptCodeGeneration(Blockly.Python, 'java');
  } else if (content.id == 'content_php') {
    Code.attemptCodeGeneration(Blockly.PHP, 'php');
  } else if (content.id == 'content_dart') {
    Code.attemptCodeGeneration(Blockly.Dart, 'dart');
  } else if (content.id == 'content_lua') {
    Code.attemptCodeGeneration(Blockly.Lua, 'lua');
  }
};
```

Figura 71 Modificación code.js 2

La última modificación a este archivo de code.js es la más larga de todas y es la inclusión del código base de la aplicación de ONOS, al ser varias páginas de código hemos decidido dejarlo en el anexo para hacer más legible la memoria.

Con todas las modificaciones a la demo de Code nos queda la siguiente interfaz:

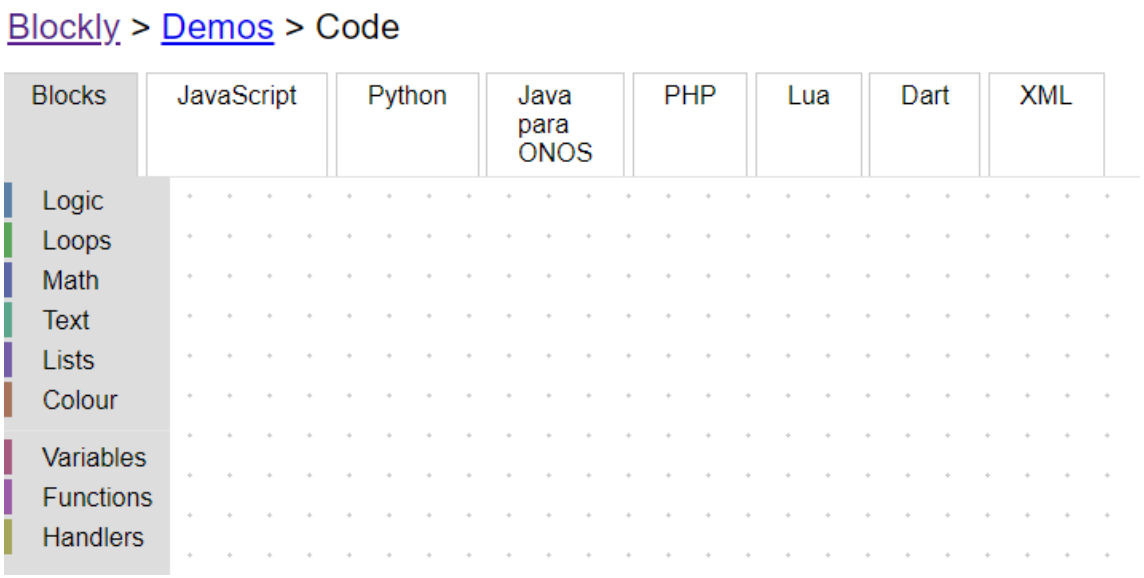


Figura 72 Interfaz visual final

Esta interfaz visual es el objetivo del trabajo, esto es lo que vería el usuario que quiere programar SDN sin necesidad de tener amplios conocimientos en el tema, de una manera muy sencilla puede abrir la pestaña de Handlers en la columna de la izquierda donde aparecen los bloques creados.

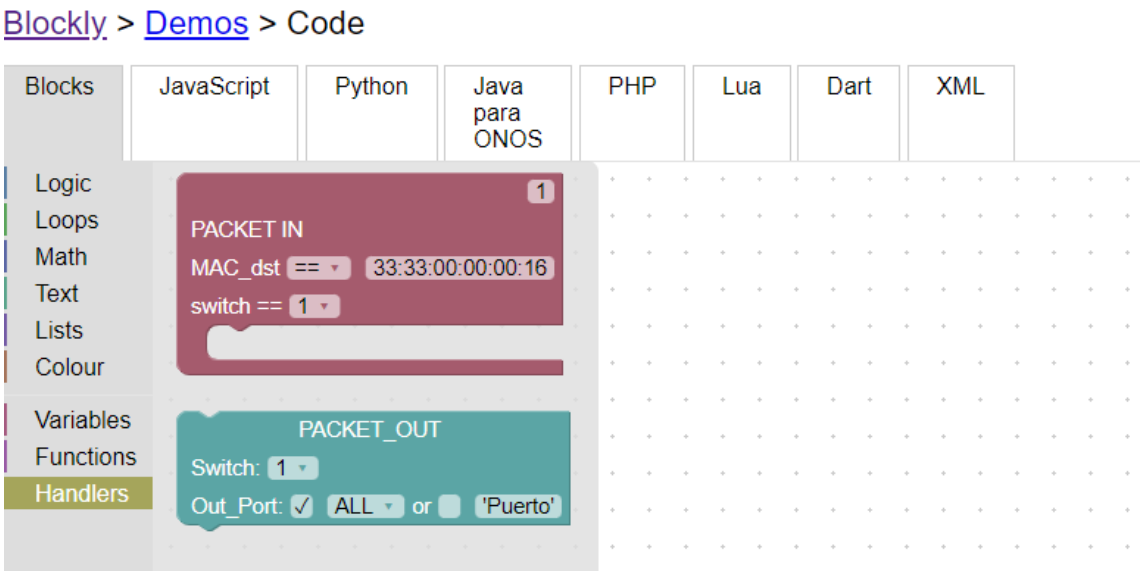
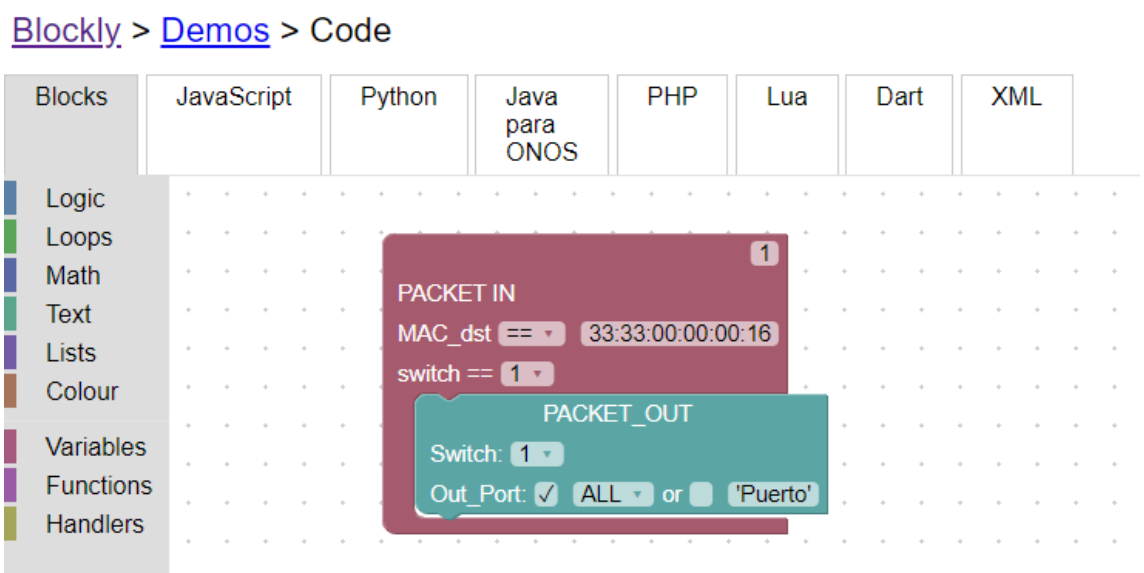


Figura 73 Interfaz visual final 2

Estos bloques los puede arrastrar a la parte central y modificar sus variables de una manera muy sencilla, una vez esta todo como quiera tan solo tiene que clickar en la opción de Java para ONOS, donde se generará el código con los bloques que ha colocado.



Este Código se puede exportar a una aplicación de ONOS como la que hemos modificado antes de OnePing y es perfectamente compilable, como veremos en el siguiente apartado.

Blockly > Demos > Code

Blocks | JavaScript | Python | Java para ONOS | PHP | Lua | Dart | XML

```
/*
 * Copyright 2015 Open Networking Foundation
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.onos.oneping;

import com.google.common.collect.HashMultimap;
import org.apache.felix.scr.annotations.Activate;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Deactivate;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.ReferenceCardinality;
import org.onlab.packet.Ethernet;
import org.onlab.packet.IPv4;
import org.onlab.packet.MacAddress;
import org.onosproject.core.ApplicationId;
import org.onosproject.core.CoreService;
import org.onosproject.net.DeviceId;
import org.onosproject.net.flow.DefaultTrafficSelector;
import org.onosproject.net.flow.DefaultTrafficTreatment;
import org.onosproject.net.flow.FlowRule;
import org.onosproject.net.flow.FlowRuleEvent;
import org.onosproject.net.flow.FlowRuleListener;
```

Figura 74 Interfaz visual final 3

5. EVALUACIÓN

Una vez desarrollado todo el proyecto vamos a demostrar su funcionamiento en este último apartado con un caso de uso:

En este ejemplo vamos a crear un controlador que intercepte un paquete que va desde el host 2 al host 1, es importante recordar la topología, especialmente tener en cuenta los puertos:

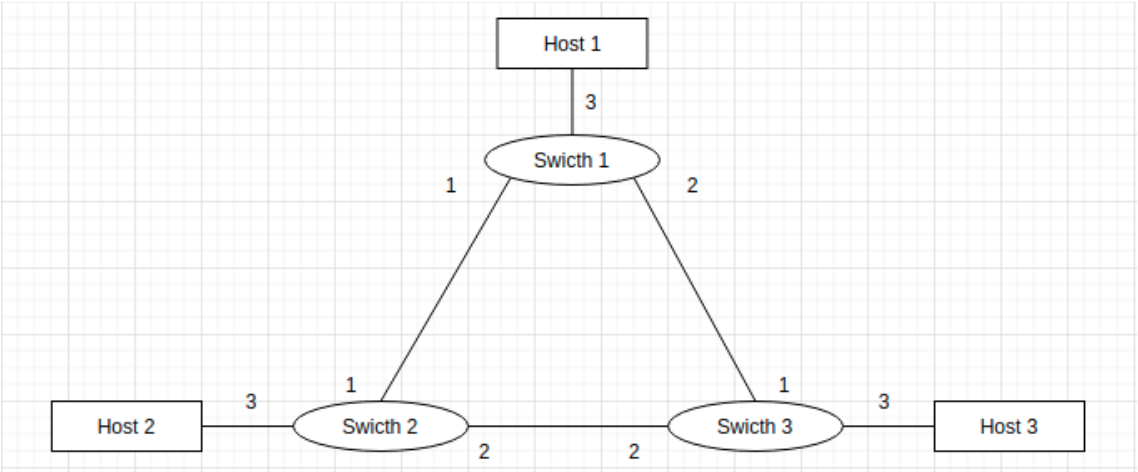


Figura 75 Toplogía del trabajo

Así pues, este paquete viajará:

- Del host 2 al Switch 2 por el puerto 3 del switch 2.
- Del switch 2 al switch 1 por el puerto 1 del switch 1.
- Del switch 1 al host 1 por el puerto 3 del switch 1.

Al utilizar pings para realizar las pruebas los paquetes hacen también el camino de vuelta.

Crearemos un conjunto de bloques: un PacketIn que pare el paquete en el switch 1 y un PacketOut que lo mande por el puerto 2, hacía el switch 3.

Lo que debería de ocurrir con estos bloques es que el TrafficSelector del PacketIn al coincidir la MAC de destino con la MAC que introducimos en el bloque debe pararlo, una vez que esto ocurre se ejecutará el TrafficTreatment (PacketOut) por el puerto que hemos seleccionado.

Veamos primero que pasa sin instalar nuestra aplicación, obviamente ya hemos ejecutado ONOS, Mininet y hemos activado las aplicaciones de fwd y OpenFlow como hemos hecho en apartados anteriores.

Comenzamos pues haciendo un ping del host 2 al host1, la opción “-c 1” hace que se mande un único paquete.

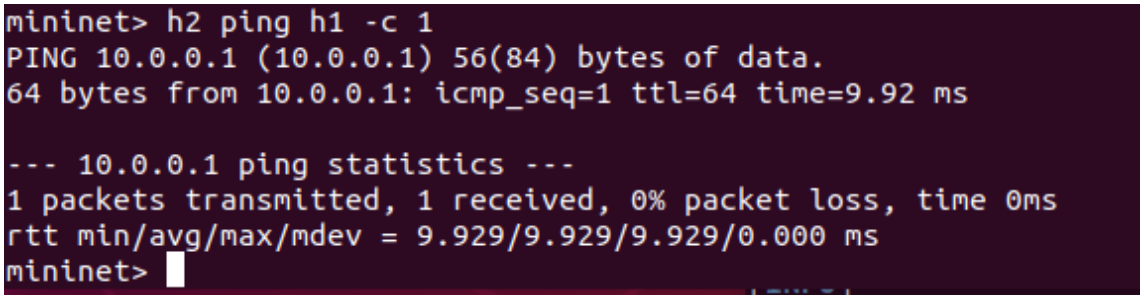


Figura 76 Ping h2 a h1

Abrimos un xterm como explicamos en el apartado anterior en el que controlamos el tráfico del switch 1, vamos a ver que ocurre en el puerto 1 usando el comando “tcpdump s1-eth1”.

Figura 77 Control tráfico switch 1 puerto 1

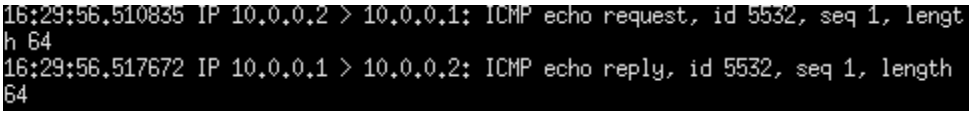


Figura 78 Paquete pasando por el switch 1 puerto 1

Podemos observar como el ping ha pasado por este puerto, tal como esperábamos, ida y vuelta.

Veamos ahora el puerto 2, de nuevo ejecutamos el comando “tcpdump” esta vez eligiendo el puerto 2:

Figura 79 Control tráfico switch 1 puerto 2

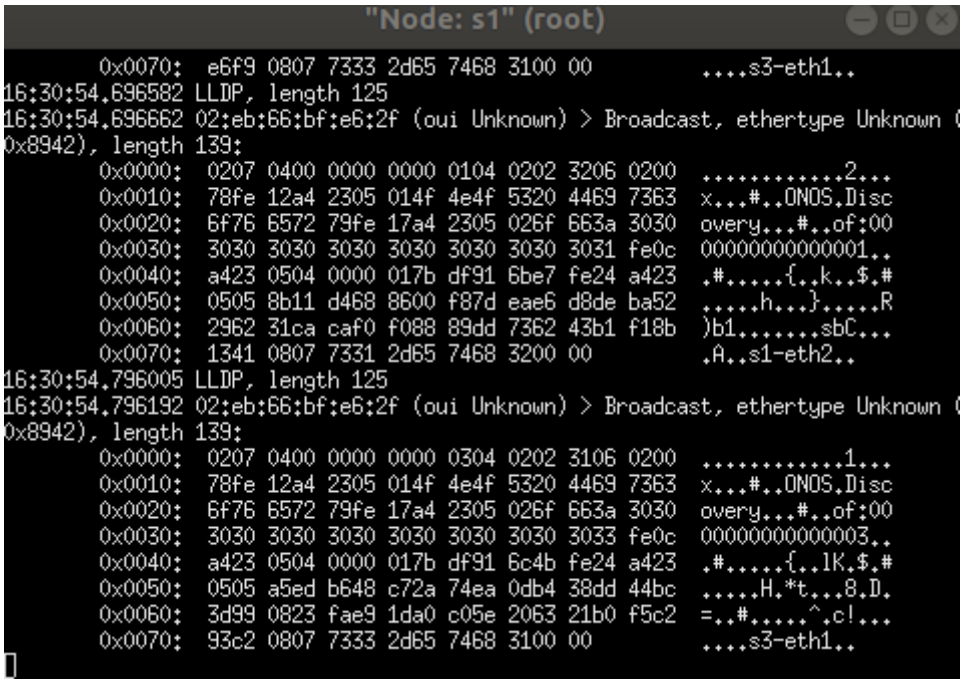


Figura 80 Tráfico switch 1 puerto 2

Como podemos ver aquí no aparece nuestro paquete, tan solo los paquetes Discovery de ONOS que pasan cada 3 segundos, tal y como esperábamos ya que este puerto no está en la ruta del paquete.

Figura 81 Tráfico switch 1 puerto 3

Por último, el puerto 3 que efectivamente por aquí pasa el ping ida y vuelta.

Ahora vamos a nuestra interfaz personalizada con Blockly y creamos un PacketIn donde la MAC destino es el Host 1 (00:00:00:00:00:01) y el switch donde queremos interceptarlo es el switch 1, recordar que este PacketIn funciona como un match, donde las condiciones que pongamos deben cumplirse para poder luego realizar la acción descrita en el PacketOut.

En el PacketOut elegimos de nuevo el Switch 1 y el puerto de salida manual, donde seleccionamos la opción de que se vaya por el puerto 2.

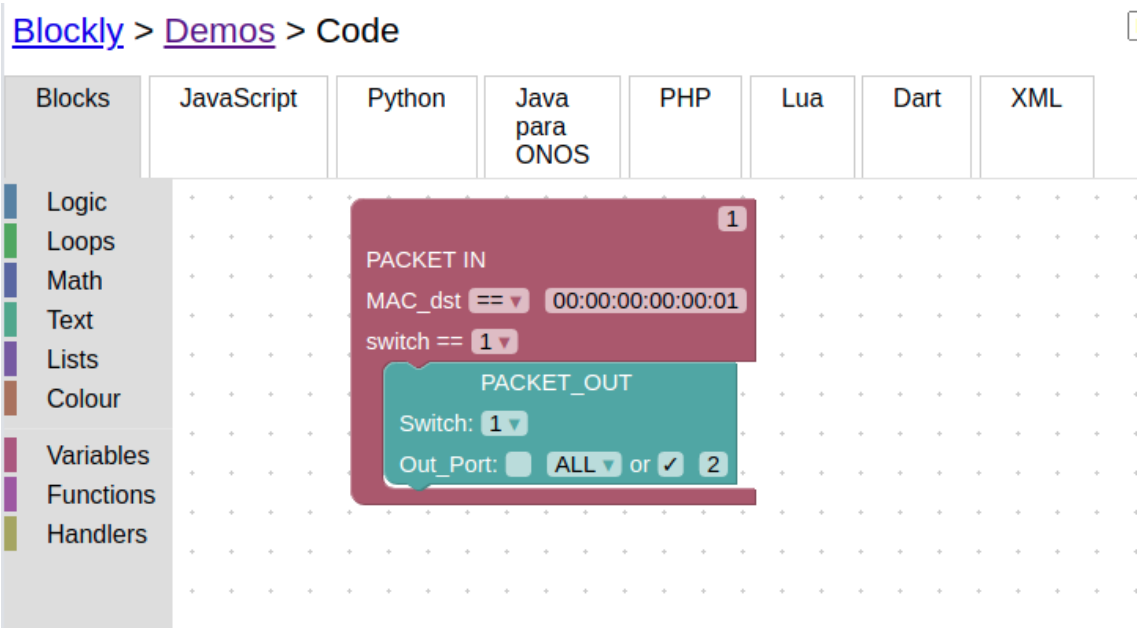


Figura 82 Creación de código con la interfaz visual

Moviéndonos al apartado de Java para ONOS donde tenemos nuestro código generado para este controlador concreto, en la siguiente foto mostramos concretamente la parte donde se ven los dos bloques que hemos creado y como coge sus parámetros.

```
}
# PACKET IN
    private void packetin(DeviceId deviceId, MacAddress src, MacAddress dst, PacketContext context) {
        TrafficSelector selector = DefaultTrafficSelector.builder()
                .matchEthSrc(src).matchEthDst(dst).build();
        if (deviceId.toString().equals("of:0000000000000001")) {
            if (dst.toString().equals("00:00:00:00:00:01")) {
                        context.treatmentBuilder().setOutput(PortNumber.portNumber(2));
                context.send();
              }
          }
          TrafficTreatment drop = DefaultTrafficTreatment.builder()
                  .setOutput(PortNumber.portNumber(2)).build();
          flowObjectiveService.forward(deviceId, DefaultForwardingObjective.builder()
                  .fromApp(appId)
                  .withSelector(selector)
                  .withTreatment(drop)
                  .withFlag(ForwardingObjective.Flag.VERSATILE)
                  .withPriority(DROP_PRIORITY)
                  .makeTemporary(TIMEOUT_SEC)
                  .add());
      }
  }
```

Figura 83 Código resultante de los bloques

Exportamos el código a la aplicación de ONOS y compilamos:

```
alvaro@tfg:~/onos/apps/oneping$ sudo mvn clean install
```

Figura 84 Compilación de la aplicación

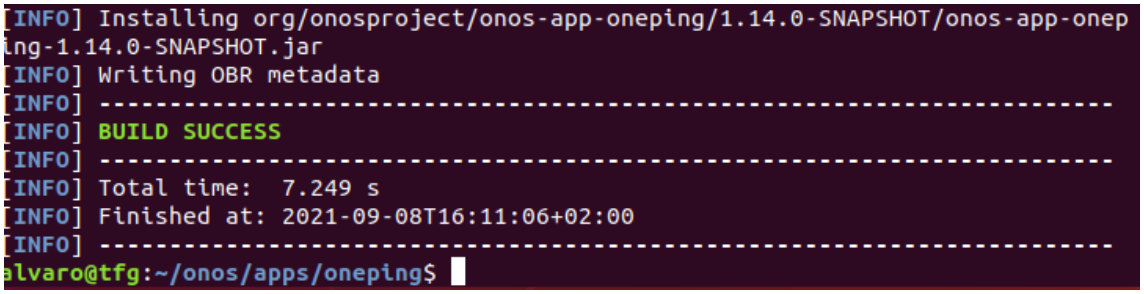


Figura 85 Resultado de la compilación

Ahora instalamos la aplicación y la activamos:

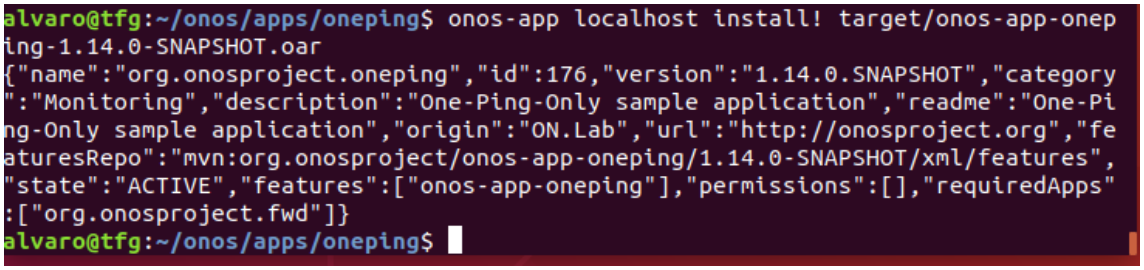


Figura 86 Instalación y activación de la aplicación

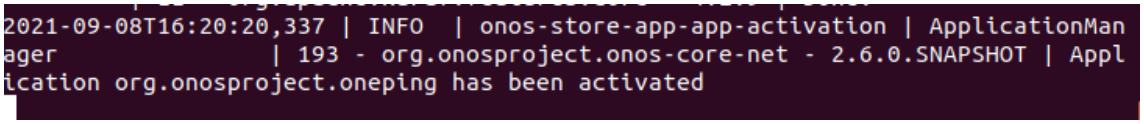


Figura 87 Resultado de la activación

Procedemos a repetir el ping de antes, ya podemos ver aquí que el paquete aparece perdido.

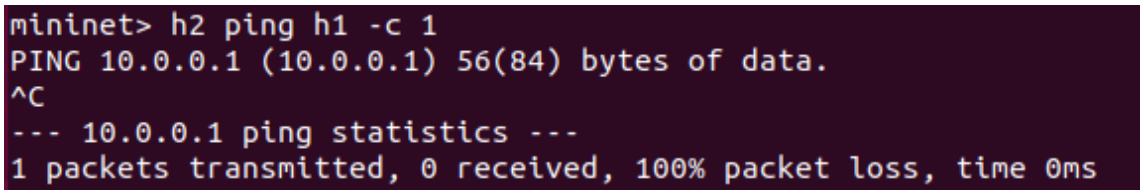


Figura 88 Ping de h2 a h1

Y tal y como podemos analizar, en este caso, el ping va del switch 2 al switch 1, donde nuestro programa de OnePing ha interceptado con el traffic selector el paquete ya que la dirección MAC de destino coincida con la que hemos seleccionado y lo ha mandado por el puerto 2, que va al switch 3, aquí es donde aparece un error que dice que no sabe dónde mandar el paquete.

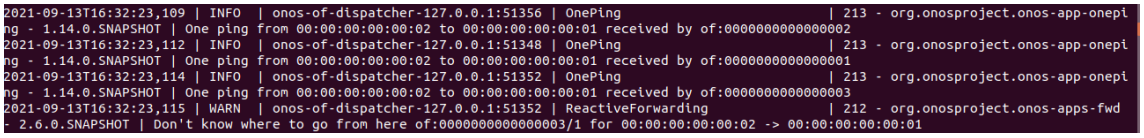


Figura 89 Resultado final

En este caso de uso hemos observado como con tan solo la interfaz visual y dos simples comandos (compilar e instalar la aplicación) hemos desarrollado un controlador de redes definidas por software sin tener ningún conocimiento del controlador a bajo nivel.

Esta interfaz visual obviamente no es muy completa, tan solo tenemos dos mensajes OpenFlow programados, no obstante, potencialmente es una herramienta muy útil, especialmente, al igual que Blockly para aquellas personas que se estén iniciando en el mundo de la programación, ya que este lenguaje es de muy alto nivel. Abstrayendo incluso la gramática de la programación.

No obstante, al igual que Blockly tiene sus limitaciones, ya que, por ejemplo, Blockly soporta Python de manera oficial y completa, sin embargo, no utilizamos este medio para programar en Python siendo mucho más sencillo que la programación normal ya que tiene limitaciones como el uso de librerías o la creación de objetos.

Es complicado evaluar como de grandes serían estas limitaciones en nuestro proyecto ya que para llegar a ellas hace falta mucho más desarrollo.

6. CONCLUSIONES Y TRABAJO FUTURO

Las SDN son un apartado de la informática con un nivel de dificultad para el desarrollo y una curva de aprendizaje bastante elevados. Este hecho se ha podido comprobar de primera mano con este trabajo, comenzamos sin conocer prácticamente nada sobre el tema y hemos acabado desarrollando un trabajo de unas 300 horas en el que no solo hemos desarrollado, sino que también hemos dedicado unas cuantas horas a la investigación de otros proyectos que tienen como objetivo el de simplificar el desarrollo de las SDN.

Este trabajo se proponía el objetivo de desarrollar una interfaz visual para el manejo de redes definidas por software para el controlador ONOS y lo hemos conseguido. Podemos debatir de la utilidad de este trabajo, ya que es obvio que para que este trabajo pueda ser usable en el día a día en el desarrollo de redes de comunicación de una persona o una empresa habría que dedicarle muchas más horas a este proyecto para que pueda competir con otras propuestas como P4 o Lucid. No obstante, este proyecto es potencialmente más interesante, ya que lleva el objetivo de simplificar el desarrollo a un nivel por encima de estos dos lenguajes, abstrayendo también la gramática de la programación.

También es cierto e innegable que la utilidad de este trabajo está completamente ligada a la cantidad de bloques que creamos y, en este caso, habiendo creado tan solo PacketIn y PacketOut, no es suficiente para la mayoría de las personas que quieran desarrollar un controlador. No obstante, es posible que para una persona que no busque nada más que eso posiblemente esta propuesta sea la más sencilla de ejecutar.

Es por esto por lo que el trabajo futuro más inmediato debería ser el de aumentar la cantidad de bloques. Se podrían crear bloques para muchos más mensajes OpenFlow, además de variables o parámetros, ya que por ejemplo los PacketIn pueden tener infinidad de parámetros por los que realizar el match, pero obviamente para este trabajo no podíamos ponerlos todos. De hecho, el objetivo de este trabajo estaba más orientado hacia hacer el primer desarrollo acercamiento de ONOS a una interfaz gráfica más que a intentar hacer algo comercial. Como hemos visto en los ejemplos, controlar el flujo de paquetes no es algo realmente complejo, pero esta muestra de concepto demuestra que podemos hacer lo que queramos con esta idea.

Hablando del trabajo futuro, también sería interesante mencionar un problema encontrado con esta interfaz visual y es el hecho de que los bloques imprimen código de manera independiente. Esto significa que, por ejemplo, no podemos en un solo programa poner dos mensajes PacketIn, ya que ambos crearían una función de nombre PacketIn() y daría error la compilación. No obstante, sería muy sencillo modificar el código para cambiar uno de los nombres de la función y que funcionase correctamente.

Finalmente, en cuanto a la dificultad de este proyecto, debido de la complejidad de las SDN, el inicio de este trabajo fue bastante complejo, tanto el entendimiento de las herramientas por separado como su unión. Mininet creo que es una herramienta muy sencilla y potente, con tan solo dedicarle unas horas ya puedes entender el programa al completo y en caso de dudas puedes consultar de manera muy sencilla su manual, todo lo contrario, ONOS, de lejos el programa al que más tiempo hemos dedicado. A pesar de estar desarrollada en Java y con un tutorial disponible, entender el despliegue de aplicaciones desarrollar en ella es bastante peculiar, teniendo bastantes añadidos que son obligatorios para su funcionamiento y muchos errores que compilaban e instalaban la aplicación, pero en tiempo de ejecución daban error en el servidor, y a pesar de desactivar la aplicación los errores seguían apareciendo y forzaban el reinicio completo del servidor. No obstante, toda esta dificultad es necesaria para ampliar el trabajo de Victoria y llevarlo a un posible despliegue real.

7. ANEXO

7.1 CÓDIGO CODE.JS

Como hemos mencionado en el apartado 4.4 al modificar el archivo code.js hemos introducido gran parte del código de la aplicación de ONOS desarrollada para que cree el código final de Blockly que luego exportamos a ONOS, debido a que el código era muy largo lo hemos decidido separar del apartado de desarrollo y dejarlo aquí en el anexo.

```
Code.attemptCodeGeneration = function(generator,
prettyPrintType) {
  var content = document.getElementById('content_' +
Code.selected);
  content.textContent = '';
  if (Code.checkAllGeneratorFunctionsDefined(generator)) {
    var code = generator.workspaceToCode(Code.workspace);
    if(content.id == 'content_javaONOS') {
      var header =
                    "/*\n"+
                    " * Copyright 2015 Open Networking
Foundation\n"+
                    " *\n"+
                    ' * Licensed under the Apache License,
Version 2.0 (the \"' + 'License' + '\");\n'+
                    " * you may not use this file except in
compliance with the License.\n"+
                    " * You may obtain a copy of the License
at\n"+
                    " *\n"+
                    " *
http://www.apache.org/licenses/LICENSE-2.0\n"+
                    " *\n"+
                    " * Unless required by applicable law or
agreed to in writing, software\n"+
                    ' * distributed under the License is
distributed on an \"' + 'AS IS' + '\" BASIS,\n'+
                    " * WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND, either express or implied.\n"+
                    " * See the License for the specific
language governing permissions and\n"+
                    " * limitations under the License.\n"+
                    " */\n"+
                    "\n"+
                    "package org.onos.oneping;\n"+
                    "\n"+
                    "import
com.google.common.collect.HashMultimap;\n"+
```

```
                    "import
org.apache.felix.scr.annotations.Activate;\n"+
                    "import
org.apache.felix.scr.annotations.Component;\n"+
                    "import
org.apache.felix.scr.annotations.Deactivate;\n"+
                    "import
org.apache.felix.scr.annotations.Reference;\n"+
                    "import
org.apache.felix.scr.annotations.ReferenceCardinality;\n"+
                    "import org.onlab.packet.Ethernet;\n"+
                    "import org.onlab.packet.IPv4;\n"+
                    "import org.onlab.packet.MacAddress;\n"+
                    "import
org.onosproject.core.ApplicationId;\n"+
                    "import
org.onosproject.core.CoreService;\n"+
                    "import org.onosproject.net.DeviceId;\n"+
                    "import
org.onosproject.net.flow.DefaultTrafficSelector;\n"+
                    "import
org.onosproject.net.flow.DefaultTrafficTreatment;\n"+
                    "import
org.onosproject.net.flow.FlowRule;\n"+
                    "import
org.onosproject.net.flow.FlowRuleEvent;\n"+
                    "import
org.onosproject.net.flow.FlowRuleListener;\n"+
                    "import
org.onosproject.net.flow.FlowRuleService;\n"+
                    "import
org.onosproject.net.flow.TrafficSelector;\n"+
                    "import
org.onosproject.net.flow.TrafficTreatment;\n"+
                    "import
org.onosproject.net.flow.criteria.Criterion;\n"+
                    "import
org.onosproject.net.flow.criteria.EthCriterion;\n"+
                    "import
org.onosproject.net.flowobjective.DefaultForwardingObjective;\n"
+
                    "import
org.onosproject.net.flowobjective.FlowObjectiveService;\n"+
                    "import
org.onosproject.net.flowobjective.ForwardingObjective;\n"+
                    "import
org.onosproject.net.packet.PacketContext;\n"+
                    "import
org.onosproject.net.packet.PacketPriority;\n"+
                    "import
org.onosproject.net.packet.PacketProcessor;\n"+
                    "import
org.onosproject.net.packet.PacketService;\n"+
                    "import org.onosproject.net.PortNumber;\n"+
                    "import
org.onosproject.net.edge.EdgePortService;\n"+
```

```
                    "import org.slf4j.Logger;\n"+
                    "import org.slf4j.LoggerFactory;\n"+
                    "\n"+
                    "import java.util.Objects;\n"+
                    "import java.util.Optional;\n"+
                    "import java.util.Timer;\n"+
                    "import java.util.TimerTask;\n"+
                    "\n"+
                    "import static
org.onosproject.net.flow.FlowRuleEvent.Type.RULE_REMOVED;\n"+
                    "import static
org.onosproject.net.flow.criteria.Criterion.Type.ETH_SRC;\n"+
                    "\n"+
                    "@Component(immediate = true)\n"+
                    "public class OnePing {\n"+
                    "\n"+
                    "    private static Logger log =
LoggerFactory.getLogger(OnePing.class);\n"+
                    "    private static final String
MSG_PINGED_ONCE =\n"+
                    '            \"' + 'One ping from {} to {}
received by {}' +'\";\n'+
                    "    private static final int PRIORITY =
128;\n"+
                    "    private static final int DROP_PRIORITY
= 129;\n"+
                    "    private static final int TIMEOUT_SEC =
60; // seconds\n"+
                    "\n"+
                    "    @Reference(cardinality =
ReferenceCardinality.MANDATORY_UNARY)\n"+
                    "    protected CoreService coreService;\n"+
                    "\n"+
                    "    @Reference(cardinality =
ReferenceCardinality.MANDATORY_UNARY)\n"+
                    "    protected FlowObjectiveService
flowObjectiveService;\n"+
                    "\n"+
                    "    @Reference(cardinality =
ReferenceCardinality.MANDATORY_UNARY)\n"+
                    "    protected FlowRuleService
flowRuleService;\n"+
                    "\n"+
                    "    @Reference(cardinality =
ReferenceCardinality.MANDATORY_UNARY)\n"+
                    "    protected PacketService
packetService;\n"+
                    "\n"+
                    "    @Reference(cardinality =
ReferenceCardinality.MANDATORY_UNARY)\n"+
                    "    protected EdgePortService
edgeService;\n"+
                    "\n"+
                    "    private ApplicationId appId;\n"+
                    "    private final PacketProcessor
packetProcessor = new PingPacketProcessor();\n"+
```

```
                    "    private final FlowRuleListener
flowListener = new InternalFlowListener();\n"+
                    "\n"+
                    "    private final TrafficSelector intercept
= DefaultTrafficSelector.builder()\n"+
                    "
.matchEthType(Ethernet.TYPE_IPV4).matchIPProtocol(IPv4.PROTOCOL_
ICMP)\n"+
                    "            .build();\n"+
                    "\n"+
                    "    private final HashMultimap<DeviceId,
PingRecord> pings = HashMultimap.create();\n"+
                    '    private final Timer timer = new
Timer(\"' + 'oneping-sweeper' + '\");\n'

      var clase =   "    @Activate\n"+
                    "    public void activate() {\n"+
                    '        appId =
coreService.registerApplication(\"' + 'org.onosproject.oneping'
+ '\",\n'+
                    '
() -> log.info(\"' + 'Periscope down.' + '\"));\n'+
                    "
packetService.addProcessor(packetProcessor, PRIORITY);\n"+
                    "
flowRuleService.addListener(flowListener);\n"+
                    "
packetService.requestPackets(intercept, PacketPriority.CONTROL,
appId,\n"+
                    "
Optional.empty());\n"+
                    '        log.info(\"' + 'Started' +
'\");\n'+
                    "    }\n"+
                    "\n"+
                    "    @Deactivate\n"+
                    "    public void deactivate() {\n"+
                    "
packetService.removeProcessor(packetProcessor);\n"+
                    "
flowRuleService.removeFlowRulesById(appId);\n"+
                    "
flowRuleService.removeListener(flowListener);\n"+
                    '        log.info(\"' + 'Stopped' +
'\");\n'+
                    "    }\n"


      var iniciar = "    private void processPing(PacketContext
context, Ethernet eth) {\n"+
                    "        DeviceId deviceId =
context.inPacket().receivedFrom().deviceId();\n"+
                    "        MacAddress src =
eth.getSourceMAC();\n"+
                    "        MacAddress dst =
eth.getDestinationMAC();\n"+
```

```
                    "        log.info(MSG_PINGED_ONCE, src, dst,
deviceId);\n"+
                    "        packetin(deviceId, src, dst,
context);\n"+
                    "    }\n"+
                    "\n"+
                    "    private boolean isIcmpPing(Ethernet
eth) {\n"+
                    "        return eth.getEtherType() ==
Ethernet.TYPE_IPV4 &&\n"+
                    "                ((IPv4)
eth.getPayload()).getProtocol() == IPv4.PROTOCOL_ICMP;\n"+
                    "    }\n"+
                    "\n"+
                    "    private class PingPacketProcessor
implements PacketProcessor {\n"+
                    "        @Override\n"+
                    "        public void process(PacketContext
context) {\n"+
                    "            Ethernet eth =
context.inPacket().parsed();\n"+
                    "            if (isIcmpPing(eth)) {\n"+
                    "                processPing(context,
eth);\n"+
                    "            }\n"+
                    "        }\n"+
                    "    }\n"+
                    "\n"+
                    "    private class PingRecord {\n"+
                    "        private final MacAddress src;\n"+
                    "        private final MacAddress dst;\n"+
                    "        PingRecord(MacAddress src,
MacAddress dst) {\n"+
                    "            this.src = src;\n"+
                    "            this.dst = dst;\n"+
                    "        }\n"+
                    "\n"+
                    "        @Override\n"+
                    "        public int hashCode() {\n"+
                    "            return Objects.hash(src,
dst);\n"+
                    "        }\n"+
                    "\n"+
                    "        @Override\n"+
                    "        public boolean equals(Object obj)
{\n"+
                    "            if (this == obj) {\n"+
                    "                return true;\n"+
                    "            }\n"+
                    "            if (obj == null || getClass()
!= obj.getClass()) {\n"+
                    "                return false;\n"+
                    "            }\n"+
                    "            final PingRecord other =
(PingRecord) obj;\n"+
```

```
                    "            return Objects.equals(this.src,
other.src) && Objects.equals(this.dst, other.dst);\n"+
                    "        }\n"+
                    "    }\n"+
                    "\n"+
                    "    private class PingPruner extends
TimerTask {\n"+
                    "        private final DeviceId
deviceId;\n"+
                    "        private final PingRecord ping;\n"+
                    "\n"+
                    "        public PingPruner(DeviceId
deviceId, PingRecord ping) {\n"+
                    "            this.deviceId = deviceId;\n"+
                    "            this.ping = ping;\n"+
                    "        }\n"+
                    "\n"+
                    "        @Override\n"+
                    "        public void run() {\n"+
                    "            pings.remove(deviceId,
ping);\n"+
                    "        }\n"+
                    "    }\n"+
                    "\n"+
                    "    private class InternalFlowListener
implements FlowRuleListener {\n"+
                    "        @Override\n"+
                    "        public void event(FlowRuleEvent
event) {\n"+
                    "            FlowRule flowRule =
event.subject();\n"+
                    "            if (event.type() ==
RULE_REMOVED && flowRule.appId() == appId.id()) {\n"+
                    "                Criterion criterion =
flowRule.selector().getCriterion(ETH_SRC);\n"+
                    "                MacAddress src =
((EthCriterion) criterion).mac();\n"+
                    "                MacAddress dst =
((EthCriterion) criterion).mac();\n"+
                    "            }\n"+
                    "        }\n"+
                    "}\n"
```

```
      content.textContent = header + clase + iniciar + code;
    } else {
      content.textContent = code;
    }
    if (typeof PR.prettyPrintOne == 'function') {
      code = content.textContent;
      code = PR.prettyPrintOne(code, prettyPrintType);
      content.innerHTML = code;
    }
  }
}
```

7.2 PRESUPUESTO

En este segundo anexo explicaremos en detalle el hardware utilizado y el presupuesto de todo el proyecto.

Comenzando por el hardware, hemos instalado Ubuntu 18.04 en un portátil con conexión a internet y con las siguientes características:

- i5-3230M
- 4GB de memoria RAM DDR3
- SSD de 240GB

El portátil tenía un precio original cercano a los 500€, pero debido a la depreciación tras varios años de uso podemos estimarlo aproximadamente en unos 200€ donde podemos encontrar propuestas similares en tiendas de segunda mano.

Como podemos observar no es necesario un hardware demasiado nuevo ni muy potente para trabajar con las herramientas que hemos utilizado. De hecho, la única herramienta que tiene unos requisitos mínimos recomendados en ONOS, los cuales son los siguientes:

- CPU de doble núcleo
- 2GB de RAM
- 10G de disco duro
- Conexión a internet

Como podemos observar, cumplimos todos los requisitos y durante el desarrollo no hemos notado que una mejora de hardware fuera a mejorar en ningún caso los tiempos de trabajo.

En cuanto a los programas utilizados, tanto ONOS como Blockly como Mininet son herramientas open source y gratuitas, tanto para uso personal como empresarial, por lo que no es necesario sumar ningún coste por su utilización.

Por último, habría que tener en cuenta la mano de obra especializada necesaria para el desarrollo del trabajo, en este caso ha sido de un solo ingeniero informático, el sueldo de un ingeniero informático recién graduado es de unos 20.000€ al año [26], un sueldo por hora aproximado de 10,50€, teniendo en cuenta que el tiempo invertido en este trabajo es de 300 horas, deja el coste de la mano de obra en 3150€.

La suma final de estos costes deja el valor de este TFG en aproximadamente 3350€.

7.3 BIBLIOGRAFIA

[1] Abstracción (informática) – Wikipedia https://es.wikipedia.org/wiki/Abstracci%C3%B3n_(inform%C3%A1tica)

[2] Redes definidas por software – Wikipedia https://es.wikipedia.org/wiki/Redes_definidas_por_software

[3] ¿Qué son las redes definidas por software? – Citrix https://www.citrix.com/es-es/solutions/app-delivery-and-security/what-is-software-defined-networking.html

[4] OpenFlow – Wikipedia https://es.wikipedia.org/wiki/OpenFlow

[5] OpenFlow https://searchdatacenter.techtarget.com/es/definicion/OpenFlow#:~:text=OpenFlow%20es%20un%20protocolo%20que,que%20le%20dice%20qu%C3%A9%20hacer

[6] OpenFlow - Flowgrammable http://flowgrammable.org/sdn/openflow/

[7] P4 https://opennetworking.org/p4/

[8] P4 https://p4.org/

[9] Mininet http://mininet.org

[10] ONOS - Wikipedia https://en.wikipedia.org/wiki/ONOS

[11] ONOS https://opennetworking.org/onos/

[12] Blockly https://developers.google.com/blockly

[13] Blockly - Wikipedia https://es.wikipedia.org/wiki/Blockly

[14] V. Noci, “Estudio de interfaz de control simplificado para plataforma SDN”, TFG, Universidad de Alcalá, 2019.

[15] Victoria Noci – Github https://github.com/VictoriaNoci/Blockly-TFG/tree/master/blockly

[16] J. Sonchack, D. Loehr, J. Rexford, D. Walker, “Lucid, A Lenguage for Control in the Data Plane”, July 2021

[17] E. Ollora, Z. Zhou, “Graph-to-P4: A P4 biolerplate code generator for parse graphs”, ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019.

[18] Instalación Mininet - http://mininet.org/download/#option-2-native-installation-from-source

[19] Instalación ONOS - https://wiki.onosproject.org/display/ONOS/Developer+Quick+Start

[20] Requisitos instalación ONOS - https://wiki.onosproject.org/display/ONOS/Installing+required+tools

[21] Topología triangular - https://github.com/ElisaRojas/miscellaneousSDN/blob/master/mininet/triangle.py

[22] Tutorial desarrollo aplicación ONOS - https://wiki.onosproject.org/display/ONOS/Template+Application+Tutorial

[23] GitHub ONOS (Aplicaciones) - https://github.com/opennetworkinglab/onos-app-samples

[24] GitHub Blockly - https://github.com/google/blockly

[25] Creación de bloques personalizados en Blockly - https://blockly-demo.appspot.com/static/demos/blockfactory/index.html

[26] Universidad Europea, ¿Cuánto gana un ingeniero informático? - https://universidadeuropea.com/blog/cuanto-gana-un-ingeniero-informatico#:~:text=En%202019%2C%20el%20salario%20de,profesionales%20mejor%20pagados%20en%20Espa%C3%B1a.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá