Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Monitorización de redes definidas por software (SDN) utilizando ONOS

Estudiante: Lara Miñones Rodríguez
Dirección: Miguel González López

A Coruña, 25 de junio de 2020.

A mi abuelo, Victoriano.

Agradecimientos

A mi tutor, Miguel González López, por avivar mi interés en este campo y por su gran ayuda y apoyo durante la realización de este trabajo.

A mis padres, María del Carmen y Darío, por creer en mí y apoyarme día a día.

A mi hermano, Brais, por estar siempre ahí y hacerme reír.

A mi madrina, Geli, por entenderme siempre y no dejar que el estrés me absorba.

A mis abuelos y familiares cercanos, pilares fundamentales de mi vida.

A Javi, por su apoyo incondicional y brindarme felicidad, haciéndome sonreír incluso en mis peores momentos.

Y a todas aquellas maravillosas personas que he conocido durante mi transcurso en la universidad, sin ellas nada hubiese sido lo mismo.

Resumen

Este proyecto se centra en el estudio de las posibilidades de monitorización de las redes definidas por software (SDN) utilizando el sistema operativo de red ONOS. Se ha utilizado Mininet como herramienta de emulación de red para recrear el comportamiento de una red SDN. Se ha desarrollado una aplicación que, haciendo uso de los servicios y aplicaciones existentes en ONOS, permite extraer información de interés acerca de la red emulada. Esta aplicación posibilita exportar la información y métricas obtenidas a la herramienta de monitorización Prometheus, almacenarlas de manera persistente en Elasticsearch y visualizarlas utilizando Kibana. Asimismo, se ha utilizado Logstash para realizar el filtrado de *logs* de ONOS y para el sondeo y procesado de su API REST, añadiendo más información a las visualizaciones creadas en Kibana. Adicionalmente, se han explorado las posibilidades de visualización de Grafana como alternativa a Kibana.

Abstract

This project is focused on studying software defined networks' (SDN) monitoring posibilities using ONOS. Mininet has been used as network emulator in order to recreate a SDN's behaviour. We developed an application that, using ONOS' services and applications, allows extracting information about the emulated network. This application allows exporting the information and metrics obtained to Prometheus, storaging it on Elasticsearch and visualizing it using Kibana. At the same time, Logstash has been used to filter ONOS' logs and poll and process its API REST, adding more information to Kibana's visualizations. Moreover, we explored Grafana's visualization posibilities as a Kibana's alternative.

Palabras clave:

- Redes definidas por software
- Monitorización
- ONOS
- Mininet
- ELK
- Grafana
- Prometheus

Keywords:

- Software defined networks
- Monitoring
- ONOS
- Mininet
- ELK
- Grafana
- Prometheus

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Desde hace unos años, las necesidades tanto de los proveedores de red como de las empresas, organizaciones y de los propios usuarios han aumentado significativamente. La computación en la nube, el *big data*, la IoT y el auge de los dispositivos móviles se han convertido en elementos imprescindibles en el día a día, conformando la motivación principal por la cual se han replanteado las diferentes arquitecturas de red tradicionales, con el fin de evolucionar a lo que hoy en día es conocido como *software-defined networking* (SDN). [5]

El objetivo de las redes definidas por software (SDNs) [1] [6] es proporcionar interfaces abiertas que permitan el desarrollo de software a través del cual se pueda controlar el comportamiento de la red de manera centralizada. Estas redes tienen una arquitectura dinámica, gestionable, adaptable y de costo eficiente, lo que las hace ideales para las altas demandas de ancho de banda y la naturaleza dinámica de las aplicaciones actuales. Esta arquitectura desacopla el control de la red del reenvío en sí de paquetes, permitiendo que el control de la red pueda ser completamente programable, de tal manera que las aplicaciones y servicios de red se abstraigan de la infraestructura de red subyacente. Según el modelo de referencia mostrado en la Figura 1.1, la red se divide en tres capas: infraestructura, control y aplicaciones.

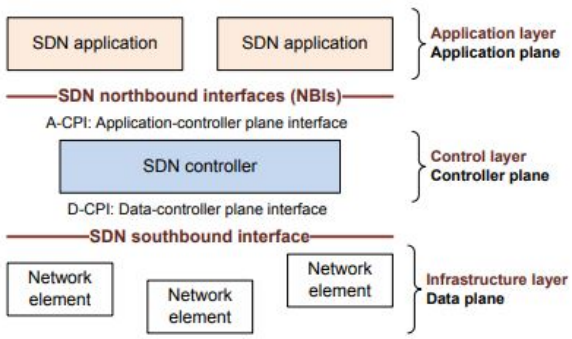


Figura 1.1: Componentes básicos de las SDN según [1].

Sin duda, las SDNs están cambiando la forma en la que se ven las redes actuales, imponiendo un nuevo paradigma en las telecomunicaciones. Las perspectivas que se avistan en cuanto a la gestión y al control del comportamiento de la red con el despliegue de SDNs son de gran interés para los operadores de servicios de telecomunicaciones. El interés existente hacia las SDNs tiene fundamentos bien justificados, tales como la posibilidad de asignación de recursos de forma dinámica o la facilitación de la administración y control de las redes de manera centralizada.

1.1 Objetivos

El objetivo principal de este proyecto es el estudio, emulación y monitorización de redes definidas por software. Para ello, se ha considerado el siguiente desglose de objetivos parciales:

- Emulación de una red SDN.
- Desarrollo de una aplicación en ONOS (Open Network Operating System) que permita obtener la información y métricas necesarias para la monitorización de la red.
- Almacenamiento de la información y métricas obtenidas.
- Creación de *dashboards* que permitan visualizar dicha información y métricas para realizar una monitorización completa de la red.

1.2 Solución planteada

La solución planteada y la interacción de las herramientas utilizadas se ilustra en la Figura 1.2. En la Sección 1.3 describimos con más detalle estas herramientas.

La emulación de la red ha sido realizada mediante un *script* escrito en Python utilizando la herramienta de emulación de redes Mininet. En dicho *script* se indica que es ONOS quien toma el rol de controlador SDN sobre la red emulada.

La aplicación para la obtención de información y métricas de red (indicada como "Prometheus Exporter" en la Figura 1.2) ha sido desarrollada en Java sobre ONOS, teniendo de este modo acceso a dicha información de forma sencilla. La aplicación recoge las métricas de la red, las formatea y las exporta a la herramienta de monitorización Prometheus. Para visualizar los datos almacenados en Prometheus, utilizamos Grafana.

Dado que Prometheus solo permite almacenar las métricas durante un período corto de tiempo, utilizamos Elasticsearch para asegurar la persistencia de los datos. Metricbeat es la

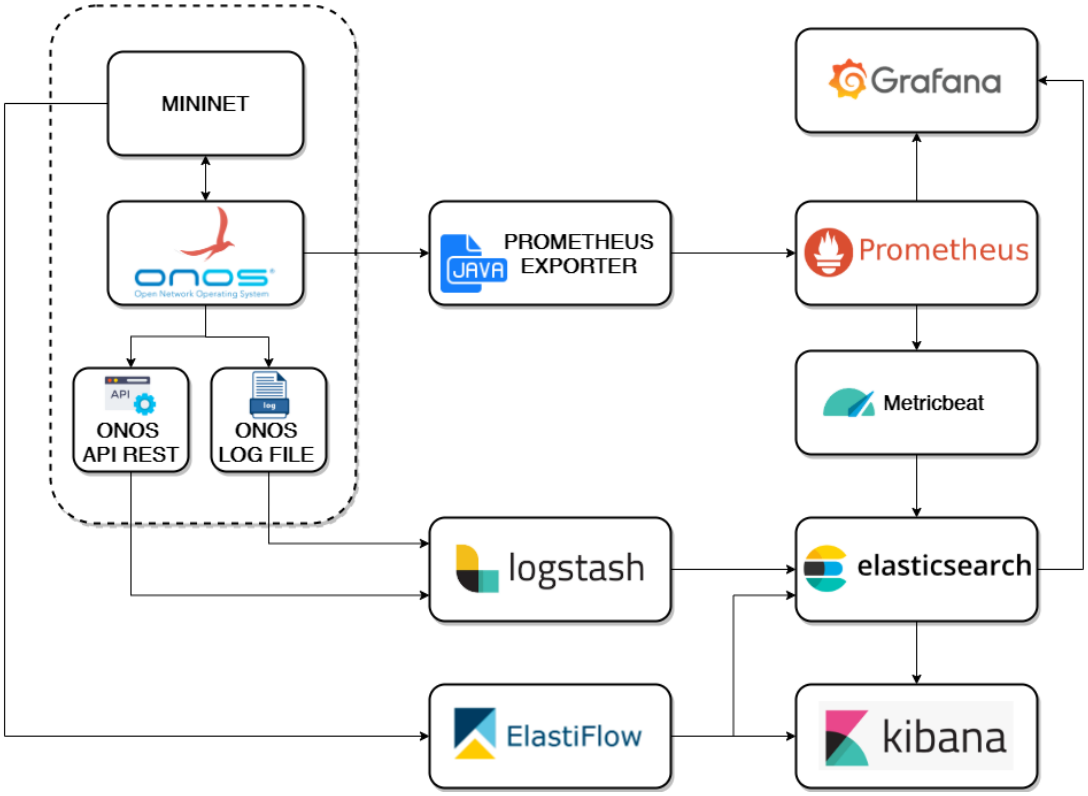


Figura 1.2: Esquema general de la herramienta de monitorización.

herramienta mediante la cual enviamos las métricas almacenadas en Prometheus a Elasticsearch.

Además, utilizamos Logstash para filtrar los *logs* de ONOS y para sondear y procesar los datos de su API REST. Utilizamos el API REST de ONOS para obtener información y métricas de red adicionales a las contempladas por la aplicación Java desarrollada. Los datos obtenidos gracias al procesado realizado por Logstash se almacenan en dos nuevos índices de Elasticsearch.

Como Grafana permite la configuración de Elasticsearch como fuente de datos, hemos creado visualizaciones en esta herramienta sobre los datos almacenados en Elasticsearch, además de los almacenados directamente en Prometheus. Sin embargo, cierta información (i.e. mensajes de *logs* de ONOS) es difícil de representar utilizando Grafana. Por ello, y por la facilidad de integración con el resto de herramientas de Elastic, hemos considerado más conveniente utilizar Kibana como herramienta de visualización de los datos.

Por último, gracias a la utilización de Kibana como herramienta de visualización, hemos importado una serie de *dashboards* con información sobre los flujos que atraviesan la red

utilizando la herramienta Elastiflow. Esta herramienta obtiene la información de los flujos de paquetes que pasan a través de los *switches* virtuales emulados en Mininet y la almacena en índices de Elasticsearch.

1.3 Tecnologías y herramientas utilizadas

En esta sección se explica brevemente cada una de las herramientas y tecnologías utilizadas para cumplir los objetivos expuestos previamente.

1.3.1 Open Network Operating System (ONOS)

Open Network Operating System [7] es una herramienta *open-source* que consiste en un controlador para redes definidas por software. ONOS maneja los componentes de la red, tales como *switches* y *links*, y lanza programas software o módulos para proveer de servicios de comunicación a los equipos finales (*hosts*) y a las redes vecinas. ONOS proporciona funcionalidades análogas a un sistema operativo, incluyendo APIs y abstracciones, asignación de recursos, permisos... así como CLI, GUI y aplicaciones de sistema. A su vez, controla la red por completo, facilitando la gestión, configuración y despliegue de nuevo software, hardware o servicios.

En este trabajo se ha utilizado ONOS como controlador, además de como sistema operativo en el cual desarrollar una aplicación que permita realizar la monitorización de una SDN.
La aplicación desarrollada permite la exportación de métricas e información sobre la red SDN emulada a Prometheus, así como la exposición de las mismas vía HTTP.

1.3.2 Bazel

Bazel [8] es una herramienta de compilación y pruebas *open-source* similar a Make, Maven y Gradle. Proporciona soporte para proyectos en múltiples lenguajes y compila salidas para múltiples plataformas (Linux, macOS y Windows). Bazel soporta la compilación y pruebas de código extenso a través de varios repositorios y un gran número de usuarios.

Esta herramienta tiene una lógica sencilla, siguiendo los siguientes pasos:

1. Realiza la carga de los ficheros BUILD, en los cuales se indican las dependencias.
2. Analiza las entradas y sus dependencias, aplica las reglas de compilación especificadas y produce un grafo de acción.
3. Ejecuta las acciones de compilación en las entradas hasta que se produce la última salida de compilación.

Como todo el trabajo de compilación previo está cacheado, Bazel puede identificar y reutilizar los artefactos y recompilar o resetear solo lo que ha cambiado.

En este trabajo se utiliza Bazel como herramienta de compilación de la aplicación desarrollada, así como del propio ONOS. Esto es así dado que ONOS trabaja por defecto con Bazel como herramienta de compilación y pruebas desde que realizó su migración desde Maven.

1.3.3 Mininet

Mininet [9] es una herramienta de emulación de redes que crea una red de *hosts*, *switches*, controladores y *links* virtuales. Los *hosts* de Mininet corren el software de red estándar de Linux, y los *switches* soportan OpenFlow para realizar enrutamiento personalizado y emulación de redes definidas por software.

En este trabajo se ha utilizado Mininet para emular una red definida por software, cuya topología y comportamiento ha sido definido en un *script* utilizando la API Python de Mininet. El tráfico de dicha red también ha sido generado según las indicaciones realizadas en dicho *script*.

1.3.4 Prometheus

Prometheus [3] es un *toolkit open-source* de monitorización y alertas del sistema. Sus características más destacables son:

- Consta de un modelo de datos multi-dimensional, cuyos datos temporales son identificados por el nombre de la métrica y pares clave-valor.
- Facilita la consulta de datos a través del lenguaje PromQL.
- No depende de almacenamiento distribuido, cada nodo del servidor es autónomo.
- Realiza la recuperación de series temporales a través de un modelo de extracción sobre HTTP.
- El descubrimiento de los *targets* se realiza a través de un servicio de descubrimiento o mediante configuración estática.

En este trabajo se ha utilizado Prometheus para realizar la recolección, extracción y almacenamiento temporal de las métricas generadas en ONOS durante la emulación de una red SDN.

1.3.5 Elasticsearch

Elasticsearch [10] es un motor de búsqueda y analítica distribuido. Permite realizar búsquedas y análisis en tiempo real para cualquier tipo de datos y almacenar e indexar datos de manera eficiente y rápida, independientemente del formato de los datos. A su vez, permite recoger y agregar información para descubrir patrones en los datos.

En este trabajo se ha utilizado Elasticsearch para almacenar las métricas de manera persistente, ya que Prometheus solo las almacena durante un período corto de tiempo.

1.3.6 Logstash

Logstash [11] es un motor de recolección de datos *open-source* con capacidad de *pipelining* en tiempo real. Permite unificar y normalizar datos de distintas fuentes dinámicamente. Además, realiza limpieza de datos para su posible utilización en casos de uso posteriores.

En este trabajo se ha utilizado Logstash para realizar el filtrado de los *logs* de ONOS y para el sondeo y procesado de los datos del API REST de ONOS, además de la creación de índices en Elasticsearch para almacenar dicha información.

1.3.7 Metricbeat

Metricbeat [12] es un componente de Elastic que permite recolectar métricas del sistema operativo y de los servicios corriendo en el sistema o servidor. Metricbeat recoge métricas y estadísticas y las envía a la salida deseada, por ejemplo Elasticsearch o Logstash.

En este trabajo se utiliza Metricbeat para enviar a Elasticsearch las métricas almacenadas en Prometheus de manera temporal.

1.3.8 Kibana

Kibana [13] es una herramienta *open-source* que permite visualizar y explorar los datos almacenados en Elasticsearch y navegar en la pila ELK (conjunto de las herramientas Elasticsearch, Logstash, Kibana y Beats).

Tanto si se es usuario o administrador, Kibana hace que sus datos sean procesables al proporcionar tres funciones clave:

- Una plataforma *open-source* para el análisis y la visualización de datos.
- Una interfaz de usuario para manejar la pila ELK. Controlar las configuraciones de seguridad, asignar roles a usuarios, hacer *snapshots*, etc.

- Un *hub* centralizado para soluciones de Elastic.

En este trabajo se ha utilizado esta herramienta para la creación de *dashboards* que permitan visualizar los datos que se han almacenado en Elastisearch, es decir, las métricas exportadas a Prometheus y los datos procesados por Logstash.

1.3.9 Elastiflow

Elastiflow [14] es una herramienta *open-source* que proporciona información sobre los flujos que atraviesan la red y aporta una serie de *dashboards* fácilmente importables a Kibana. Soporta NetFlow v5/v9, sFlow e IPFIX (Internet Protocol Flow Information Export).

En este trabajo se utiliza Elastiflow para añadir *dashboards* a Kibana y mostrar así información útil sobre la red emulada.

1.3.10 Grafana

Grafana [15] es una herramienta *open-source* que permite realizar consultas, visualizar y crear alertas acerca de datos almacenados en diferentes fuentes de datos. Además de crear, explorar y compartir *dashboards*.

En este trabajo se ha utilizado Grafana como herramienta para la creación de un *dashboard* inicial sobre los datos de la red. Se optó por comenzar con esta herramienta debido a su sencillez, obteniendo un primer *dashboard* con el que comprobar el correcto funcionamiento de la aplicación desarrollada.

Capítulo 2

Metodología, seguimiento y costes

2.1 Metodología

Para la realización de este trabajo se ha optado por una metodología incremental e iterativa, comenzando por la construcción de una aplicación sencilla a la que se le han ido añadiendo funcionalidades a lo largo del tiempo.

En la primera reunión se estableció el tema de este trabajo, así como los objetivos y fases principales del mismo, descritos en el anteproyecto. Dichas fases son las siguientes:

- Revisión del estado del arte sobre SDN.
- Estudio del sistema operativo de red ONOS.
- Estudio de las aplicaciones de ejemplo de ONOS y revisión de su código fuente.
- Desarrollo de una aplicación para la recogida de métricas.
- Inclusión de la funcionalidad de escritura en base de datos a la aplicación.
- Desarrollo de la interfaz gráfica de visualización.

2.2 Seguimiento

Se han hecho una serie de reuniones durante las diferentes fases del proyecto, de aproximadamente una a tres horas de duración, destinadas a la revisión y corrección de lo realizado durante la fase y a la decisión de los objetivos a obtener antes de la siguiente reunión.

Con el fin de facilitar el seguimiento del proyecto se creó un documento compartido con el tutor en el cual se indicaban los objetivos conseguidos y problemas encontrados. El diagrama de Gantt puede verse en las Figuras 2.1, 2.2, 2.3 y 2.4. A continuación, se exponen las diferentes iteraciones realizadas:

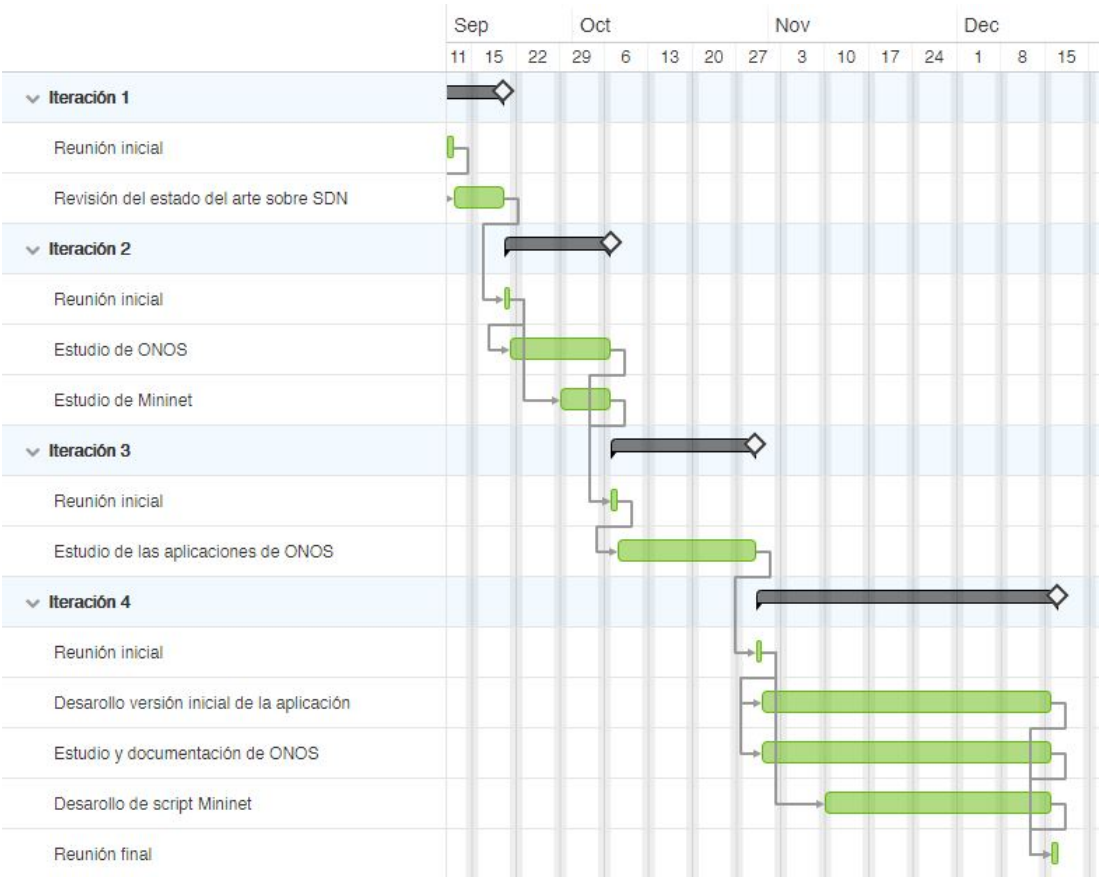


Figura 2.1: Diagrama de Gantt (1).

- **Iteración 1 (11/09/19 - 19/09/19)**: revisión del estado del arte sobre SDN.
- **Iteración 2 (20/09/19 - 06/10/19)**: estudio del sistema operativo de red ONOS y la herramienta de emulación de redes Mininet.
- **Iteración 3 (07/10/19 - 29/10/19)**: estudio de las aplicaciones de ejemplo de ONOS y revisión de su código fuente.
- **Iteración 4 (30/10/19 - 16/12/19)**: realización de las primeras versiones de la aplicación en ONOS y emulación de una red simple utilizando Mininet. Durante esta iteración se continuó el estudio de ONOS, su código y aplicaciones.
- **Iteración 5 (24/01/20 - 03/03/20)**: desarrollo de una aplicación para la recogida de métricas y exportación de las mismas a Prometheus.
- **Iteración 6 (04/03/20 - 19/03/20)**: creación del primer *dashboard* para la visualización de las métricas utilizando Grafana, modificación de la topología emulada y generación de una mayor cantidad de tráfico de red utilizando Mininet.

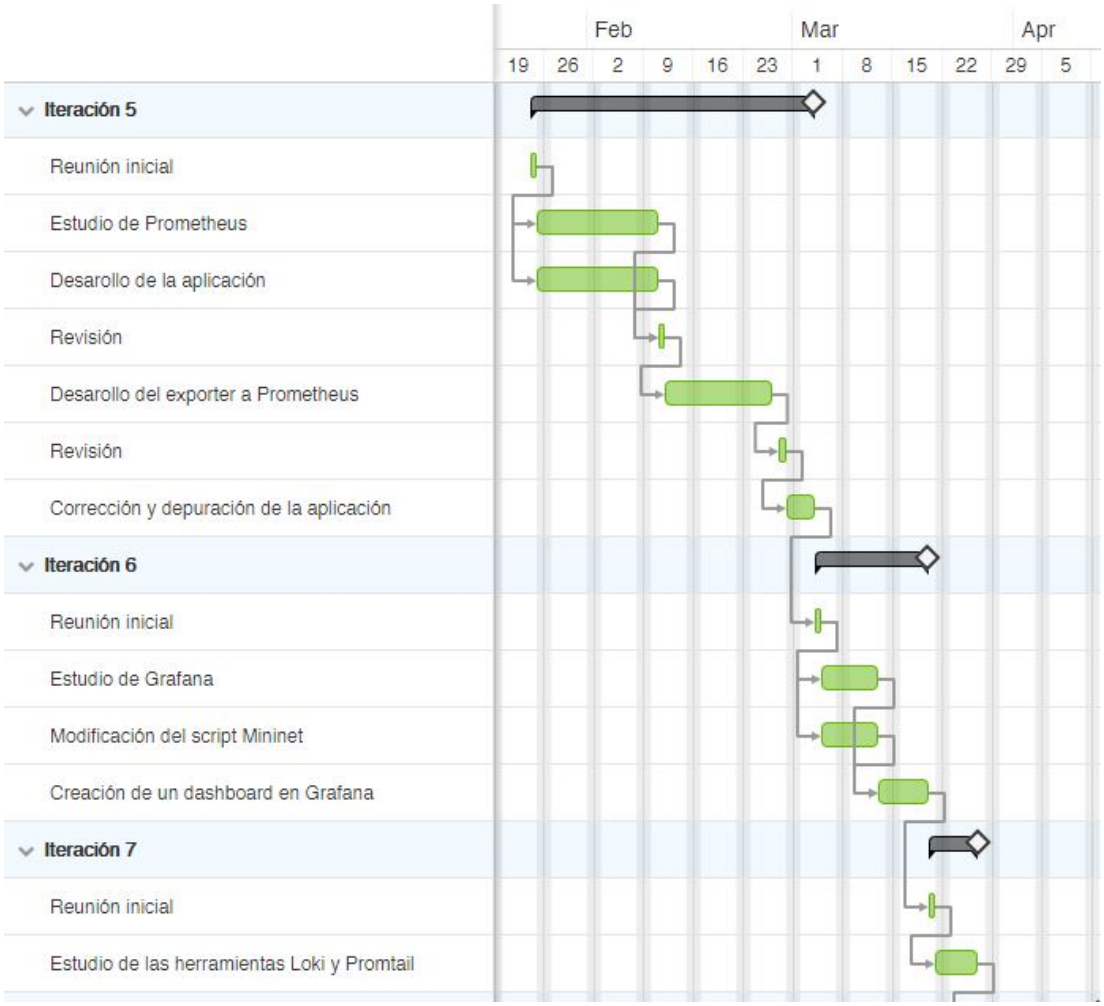


Figura 2.2: Diagrama de Gantt (2).

- **Iteración 7 (20/03/20 - 26/03/20)**: estudio, instalación y pruebas de las herramientas Loki y Promtail con el fin de filtrar el fichero de *logs* de ONOS y mostrar más información en el *dashboard* creado en Grafana. Optando finalmente por descartarlas al no obtener los objetivos deseados.
- **Iteración 8 (27/03/20 - 12/04/20)**: estudio, instalación y pruebas de la pila ELK, almacenamiento de las métricas en Elasticsearch y creación del primer *dashboard* utilizando Kibana.
- **Iteración 9 (13/04/20 - 20/04/20)**: filtrado del fichero de *logs* de ONOS utilizando Logstash y creación de un *dashboard* en Kibana para la visualización de la información obtenida de ellos.
- **Iteración 10 (21/04/20 - 29/04/20)**: sondeo y procesado de métricas obtenidas del API

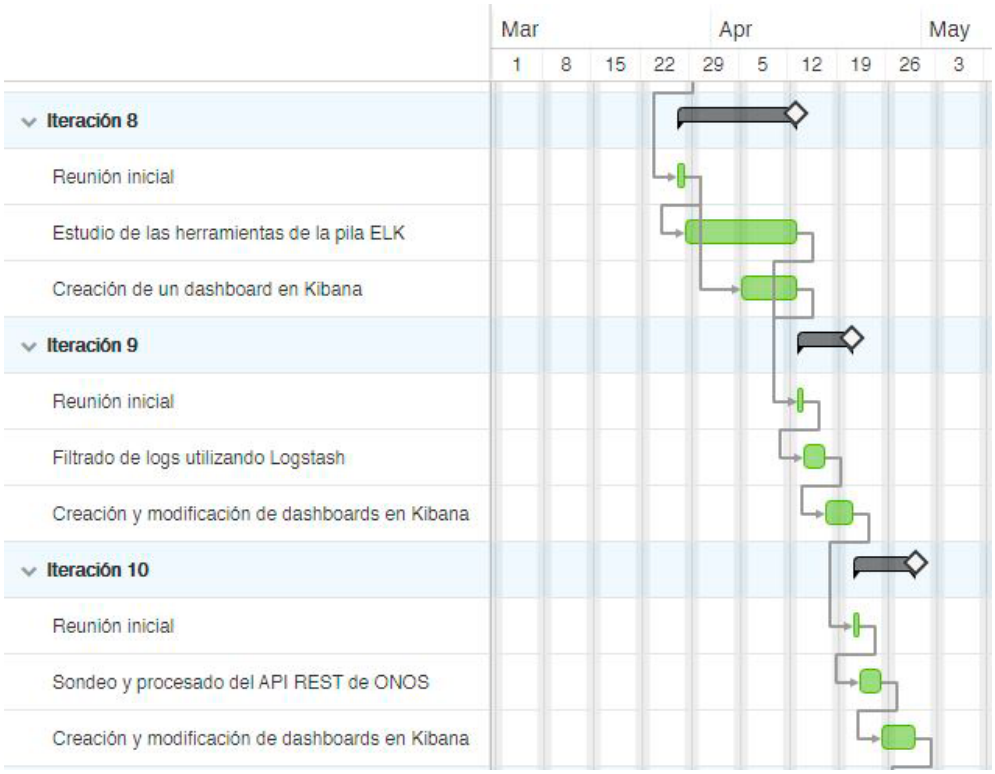


Figura 2.3: Diagrama de Gantt (3).

REST de ONOS y mejora de los *dashboards* creados en Kibana, añadiendo gráficos para visualizar las nuevas métricas obtenidas.

- **Iteración 11 (30/04/20 - 10/05/20)**: estudio e instalación de la herramienta Elastiflow para la obtención de más información acerca de los flujos que recorren la red e importación de *dashboards* en Kibana para visualizar dicha información.
- **Iteración 12 (11/05/20 - 26/06/20)**: redacción de este documento.

2.3 Costes

Se estima un trabajo de 14 horas semanales durante los primeros meses (septiembre a marzo), correspondientes a las iteraciones de la 1 a la 5, y de 28 horas semanales durante los últimos meses (marzo a junio), correspondientes a las iteraciones de la 6 a la 12.

Siguiendo las fechas indicadas en la sección anterior, se estiman 664 horas de trabajo total por parte de la alumna y 44 horas por parte del director. Suponiendo que la alumna toma el papel de analista-programador y el director de jefe de proyecto, asumimos un coste de 15€/h y 30€/h respectivamente. En total, el coste referente a los recursos humanos asciende a 11.280€.

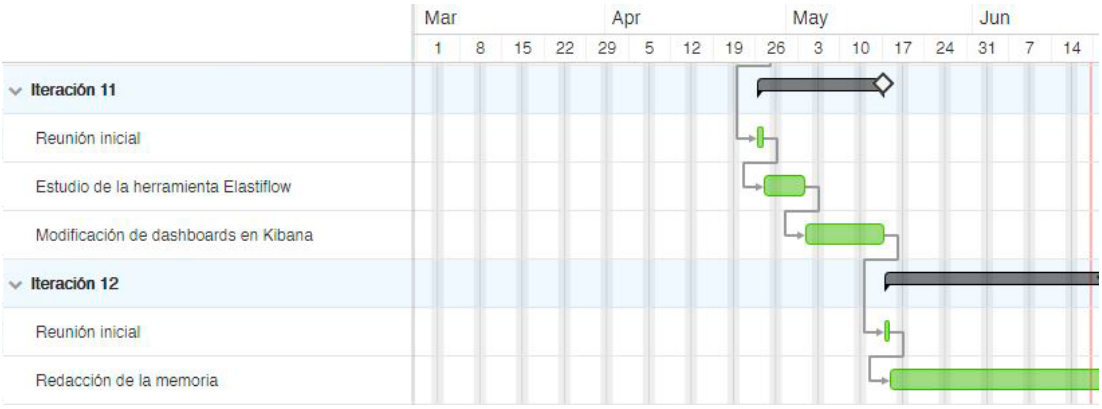


Figura 2.4: Diagrama de Gantt (4).

En cuanto a los recursos materiales y las herramientas y tecnologías utilizadas, se asume un gasto de 0€ ya que se han utilizado herramientas *open-source* y no ha sido necesaria la compra de ningún recurso material. En el Cuadro 2.1 se muestra en detalle cómo se han realizado los cálculos.

	Tiempo dedicado (h)	**Coste (€/h)**	**Coste total (€)**
Alumna	664h	15€/h	9960€
Director	44h	30€/h	1320€
		Coste total del proyecto:	**11.280€**

Cuadro 2.1: Coste del proyecto.

Capítulo 3

Emulación de una red definida por software

PARA realizar la emulación de una red definida por software se ha utilizado Mininet como herramienta de emulación de redes y ONOS como controlador SDN.

Tras haber realizado la instalación y configuración de ambas herramientas, el primer paso consiste en el desarrollo de un *script* que permita emular la red deseada. Se ha optado por una topología de red simple, que permita generar información y tráfico de red suficiente para la monitorización de la misma, sin dificultar dicho propósito.

3.1 Mininet

Mininet [9] es la herramienta de emulación de redes utilizada en este trabajo. Los *hosts* de Mininet corren el software de red estándar de Linux, y los *switches* soportan OpenFlow para realizar enrutamiento personalizado y redes definidas por software.

Las redes Mininet corren código real incluyendo tanto aplicaciones de red estándar de Unix/Linux como el kernel real de Linux y la pila de red (incluyendo cualquier extensión del kernel disponible siempre y cuando sean compatibles con los espacios de nombres de la red). Por ello, el usuario puede desarrollar y probar código en Mininet, para un controlador OpenFlow, modificar *switches* o *hosts*, realizar pruebas de evaluación del comportamiento de una red, despliegue de una red, etc. Lo más importante es que un diseño de red que funciona correctamente en Mininet normalmente podrá ser trasladado directamente a *switches* hardware.

Mininet utiliza virtualización basada en procesos para lanzar muchos *hosts* y *switches* en un único kernel de sistema operativo. Mininet puede crear un kernel o espacio de usuario para los *switches* OpenFlow y controladores y así tomar control de los *switches*, y *hosts* con el

fin de realizar la comunicación a través de la red. Mininet conecta *switches* y *hosts* utilizando pares de *virtual ethernet* (veth). A pesar de que Mininet depende del kernel de Linux, en un futuro es posible que soporte otros sistemas operativos con virtualización basada en procesos, tales como Solaris o FreeBSD. El código de Mininet está escrito casi por completo en Python a excepción de una utilidad escrita en C.

Mininet combina muchas características de los mejores emuladores, entornos de pruebas hardware y simuladores. Comparado con aproximaciones que realizan una virtualización completa del sistema, Mininet tiene un arranque más rápido, soporta un número de dispositivos mucho mayor y proporciona un mayor ancho de banda, normalmente 2 Gbps en total. Es fácil de instalar e incluso proporciona una máquina virtual pre-empaquetada que corre tanto en VMware como en VirtualBox para sistemas operativos Mac, Windows y Linux con herramientas OpenFlow 1.0 instaladas. Comparado con entornos de pruebas hardware, Mininet es mucho menos costoso, siempre está disponible y puede reconfigurarse y re-lanzarse rápidamente. Comparado con otros emuladores, Mininet corre código real y sin modificar, incluyendo aplicaciones, kernel de sistemas operativos y controladores. Puede conectarse fácilmente a redes reales y ofrece un comportamiento interactivo.

Sin embargo, Mininet presenta limitaciones como la imposibilidad de que las redes emuladas excedan la CPU o ancho de banda disponible en un único servidor, o no poder correr *switches* OpenFlow o aplicaciones no compatibles con Linux.

3.2 Escenario base

La red emulada está compuesta por cuatro *switches*, conectados de forma lineal entre sí, con dos *hosts* conectados a cada uno de ellos, tal y como puede verse en la Figura 3.1. Todos los *switches* que forman la topología de la red son de tipo *Open Virtual Switch* [16], en los cuales se han activado los protocolos OpenFlow y NetFlow.

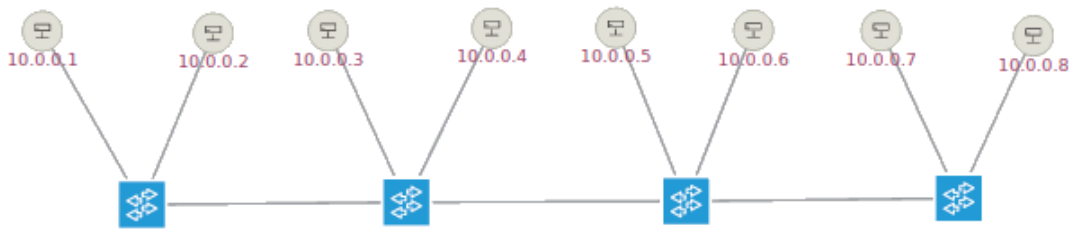


Figura 3.1: Topología de la red.

3.3 Protocolo OpenFlow

OpenFlow [17] es el protocolo principal del plano de control de las redes definidas por software. Este protocolo opera en escalas temporales cercanas al tiempo que tarda un paquete de datos en volver a su emisor tras haber pasado por el destinatario del mismo, típicamente conocido en inglés como *round-trip time* (RTT). Es decir, suele operar en milisegundos. OpenFlow es mayormente utilizado para programar tablas de flujo que controlen el reenvío de paquetes utilizando el modelo *match-action*.

Para que este protocolo pueda ser utilizado por los *switches* de la topología, además de haber activado el protocolo OpenFlow para cada uno de ellos en el *script* desarrollado para la emulación de la red en Mininet, es necesario activar la aplicación *openflow* en ONOS.

3.4 Protocolo NetFlow

NetFlow [18] es un protocolo que puede ser activado en los *switches* para recolectar información detallada acerca de los flujos IP de una red, útil para realizar la monitorización de la misma. Este protocolo permite caracterizar el tráfico IP y ver cómo y dónde fluye la información crítica para la disponibilidad y rendimiento de la red.

Monitorizar los flujos de tráfico IP mejora la capacidad de planificación y asegura que los recursos sean utilizados de manera apropiada. Además, ayuda a determinar dónde aplicar *Quality of Service* (QoS), optimizar el uso de recursos y detectar ataques de denegación de servicio (DoS) entre otros.

NetFlow ayuda a encontrar solución a muchos de los problemas con los que los profesionales IT deben lidiar a la hora de gestionar una red, tales como el análisis de nuevas aplicaciones y su impacto en la red; la reducción del pico de tráfico máximo en redes WAN (Wide Area Network); la resolución de problemas de red; la detección de tráfico no autorizado; la detección de anomalías o la validación de parámetros de *Quality of Service* (QoS).

NetFlow construye los flujos IP basándose en los siguientes atributos de cada uno de los paquetes reenviados por un *router* o *switch*: dirección IP origen, dirección IP destino, puerto origen, puerto destino, tipo de protocolo de capa 3 y tipo de servicio e interfaz del *router* o *switch*. Todos los paquetes con la misma dirección origen/destino, mismos puertos origen/-destino, mismos protocolos de interfaz y mismo tipo de servicio se agrupan en un flujo.

Para que este protocolo pueda ser utilizado por los *switches* de la topología debe habilitarse en cada uno de ellos en el *script* desarrollado para la emulación de la red en Mininet.

3.5 Despliegue y emulación de la red

Como se ha indicado en apartados anteriores, se ha desarrollado un *script* en Python para permitir el despliegue y emulación de la red.

En primer lugar, se especifican los dispositivos de la red y sus conexiones utilizando las siguientes funciones:

- Para añadir un *switch* a la topología, siendo X el número asociado a dicho *switch*:

```
sX = self.addSwitch('sX')
```

- Para añadir un *host* a la topología, siendo X el número asociado a dicho *host*:

```
hX = self.addHost('hX')
```

- Para añadir un enlace entre dos dispositivos de la red:

```
self.addLink(dispositivo1,dispositivo2)
```

Una vez definida la topología de la red, se debe indicar la dirección IP del controlador SDN, el tipo de *switch* utilizado en la red y los protocolos que deben activarse en cada uno de ellos. En este caso se ha utilizado el controlador SDN de ONOS, *switches* de tipo *Open Virtual Switch* y se han activado los protocolos OpenFlow y NetFlow en todos los *switches*. Para ello se han utilizado las siguientes funciones:

- Para indicar la dirección IP del controlador SDN, el tipo de *switch* y la activación del protocolo OpenFlow en su versión 1.3:

```
net = Mininet( topo=topo, controller=partial( RemoteController,
ip='xxx.xxx.xxx.xxx' ),  switch=partial( OVSSwitch, protocols='
OpenFlow13' ) )
```

- Para realizar el despliegue de la red:

```
net.start()
```

- Para activar NetFlow en los *switches* de la red, siendo X el número asociado al *switch*:

```
sX = net.getNodeByName('sX')
sX.vsctl('set Bridge sX netflow=@id -- --id=@id create NETFLOW
targets=\"127.0.0.1\:2055\"')
```

- Para el volcado de las conexiones de los *hosts*:

```
dumpNodeConnections(net.hosts)
```

Llegados a este punto, ya se habría desplegado la topología contra el controlador SDN con el tipo de *switches* y protocolos deseados. Tras ello se ha hecho uso de una serie de funciones que permiten generar tráfico de red:

- Se ha generado tráfico ICMP y probado la conectividad de la red utilizando el siguiente comando para realizar un ping desde todos los *hosts* a todos los *hosts*:

  ```
  net.pingAll()
  ```

- Se ha generado tráfico TCP utilizando la función *iperf* desde todos los *hosts* a todos los *hosts*. A continuación se muestra un ejemplo del uso de dicha función para generar tráfico TCP con origen el *host* "h1" y destino el *host* "h2":

  ```
  net[ 'h2' ].cmd('iperf -s -p 6000 &')
  net[ 'h1' ].cmd('iperf -c 10.0.0.2 -p 6000')
  ```

- De igual manera que con el tráfico TCP, se ha generado tráfico UDP utilizando la función *iperf* desde todos los *hosts* a todos los *hosts*. A continuación se muestra un ejemplo del uso de dicha función para generar tráfico UDP con origen el *host* "h1" y destino el *host* "h2":

  ```
  h1, h2 = net.getNodeByName('h1', 'h2')
  net.iperf( ( h1, h2 ), l4Type='UDP' )
  ```

Tras la generación del tráfico de red, se utiliza la siguiente función para permitir el acceso a la línea de comandos de Mininet desde la terminal en la que se lance este *script*:

```
CLI(net)
```

Finalmente, se para la topología de la red utilizando la siguiente función:

```
net.stop()
```

En el Capítulo B se adjunta el código completo de este script.

Una vez desarrollado el *script*, debe ejecutarse ONOS como controlador SDN y, tras ello, debe lanzarse el *script*. Para ejecutar ONOS ha de abrirse una terminal en la carpeta ONOS_ROOT y ejecutar el siguiente comando:

```
bazel run onos-local -- clean debug
```

Una vez el comando anterior haya finalizado, debe abrirse una nueva terminal en la carpeta en la que resida el *script* desarrollado y ejecutar el siguiente comando, siendo "script.py" el nombre con el que se ha guardado el *script*:

```
sudo python script.py
```

En la Figura 3.2 se muestra un ejemplo de la ejecución del *script*. Puede observarse como se añaden todos los dispositivos y sus conexiones, el lanzamiento del controlador y de los *switches*, el volcado de las conexiones de los *hosts* y el principio de la generación de tráfico de red.

No se ha considerado añadir una captura de la salida completa de la ejecución del *script* debido a la gran extensión de información mostrada al generar el tráfico entre todos los *hosts* de la red.

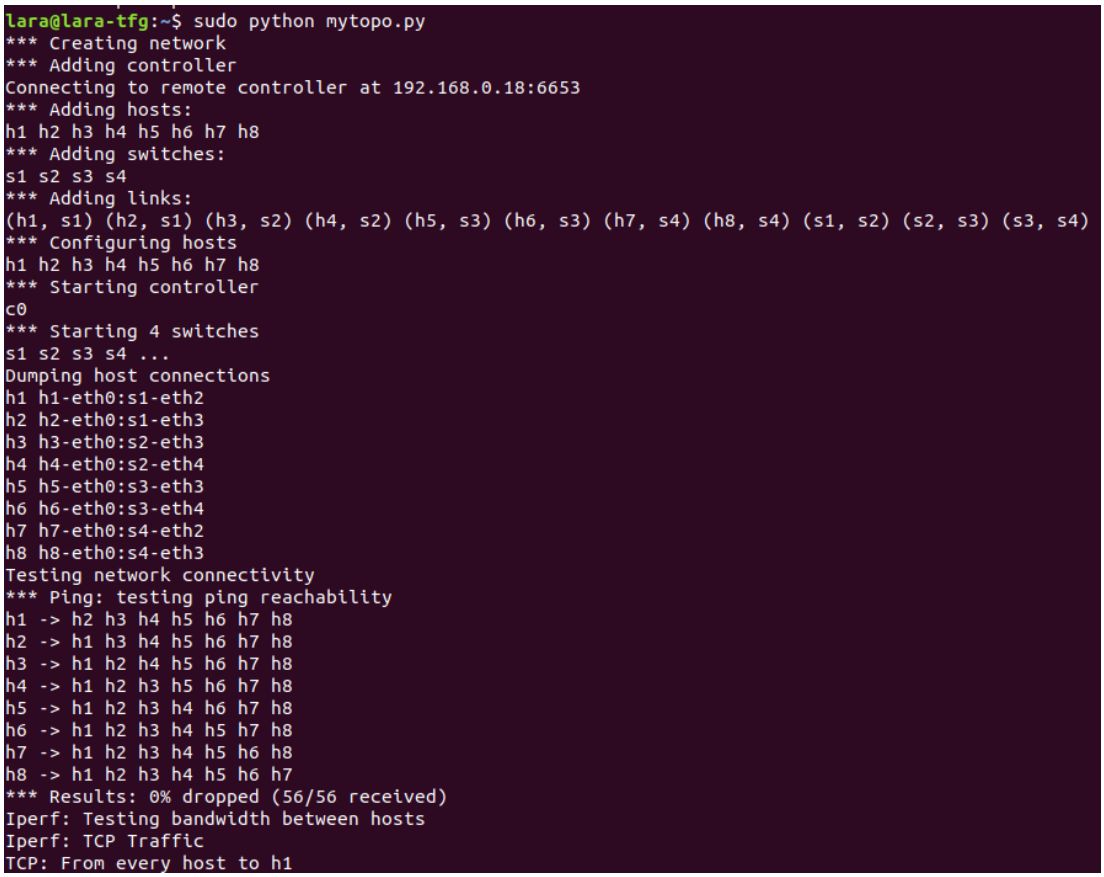


Figura 3.2: Ejecución del *script* de simulación de la red.

Capítulo 4

Desarrollo de una aplicación de monitorización en ONOS

Uno de los objetivos principales de este proyecto es la creación de una aplicación en ONOS que permita obtener la información y métricas necesarias para realizar la monitorización de una red definida por software. Para ello, ha tenido que realizarse un estudio previo de ONOS y de la estructura de sus aplicaciones, así como de las funcionalidades ofrecidas por cada una de estas.

En las diferentes secciones de este capítulo se explica en detalle tanto la arquitectura de ONOS, como la estructura general de sus aplicaciones y el desarrollo de esta aplicación en concreto.

4.1 ONOS

Open Network Operating System [19] es un controlador SDN *open-source* creado principalmente para el desarrollo de soluciones SDN y NFV (Network Function Virtualization). ONOS controla los dispositivos y componentes de la red y proporciona servicios de comunicación a los *hosts* y a las redes vecinas. Es un proyecto multi-módulo cuyos módulos son controlados por *bundles* OSGi.

ONOS fue diseñado con cuatro metas en mente:

1. **Modularidad de código**, siendo posible introducir nuevas funcionalidades como unidades auto-suficientes.
2. **Configurabilidad**, siendo posible cargar y descargar diferentes características, tanto antes de lanzar ONOS como durante su ejecución. ONOS asegura esto gracias al uso de Apache Karaf y el *framework* OSGi.

3. **Separación de conceptos**, dejando claro los límites entre subsistemas para facilitar la modularidad.

4. **Agnosticismo de protocolos**. ONOS y sus aplicaciones no deberían de estar ligados a librerías de protocolos específicas o implementaciones.

4.2 Arquitectura

ONOS [2] está formado por una serie de componentes y subsistemas, agrupados por capas de funcionalidades. La arquitectura de ONOS (Figura 4.1) se divide en: aplicaciones (*apps*), *Northbound (consumer) API*, *Core*, *Southbound (provider) API*, *Providers*, *Protocols* y *Network Elements*.

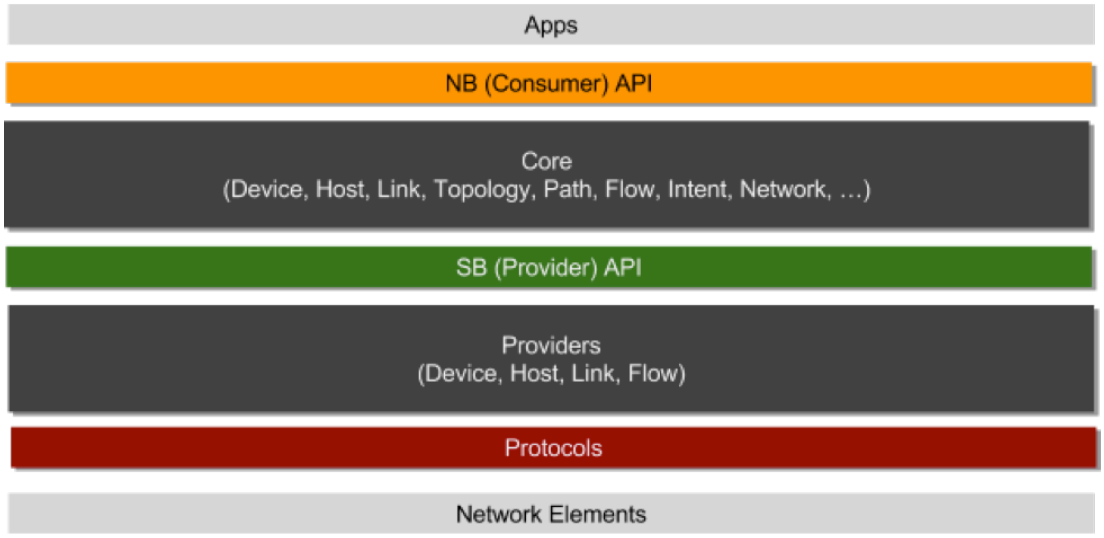


Figura 4.1: Arquitectura de ONOS según [2].

4.2.1 Core

El Core de ONOS está formado por servicios y subsistemas. Un servicio es una unidad de funcionalidad compuesta por múltiples componentes diferenciados que conforman una pila *software*. Llamamos subsistema a la colección de componentes que forman un servicio. ONOS define los siguientes subsistemas primarios en su *Core*:

- *Device Subsystem*: controla el inventario de los dispositivos de la red.
- *Link Subsystem*: gestiona el inventario de *links*.
- *Host Subsystem*: controla el inventario de *hosts* finales y sus localizaciones en la red.

- *Topology Subsystem*: gestiona *snapshots* del grafo de la red ordenadas temporalmente.
- *Path Subsystem*: computa y encuentra caminos entre los dispositivos de la red o *hosts* utilizando el grafo de la *snapshot* más reciente.
- *FlowRule Subsystem*: controla el inventario de reglas *match/action* instaladas en los dispositivos de la red y proporciona métricas de flujo.
- *Packet Subsystem*: permite que las aplicaciones escuchen los paquetes de datos recibidos por los dispositivos de la red y que envíen paquetes de datos a la red a través de uno o más dispositivos.

La Figura 4.2 ilustra los subsistemas que forman parte de ONOS actualmente. Dichos sub-

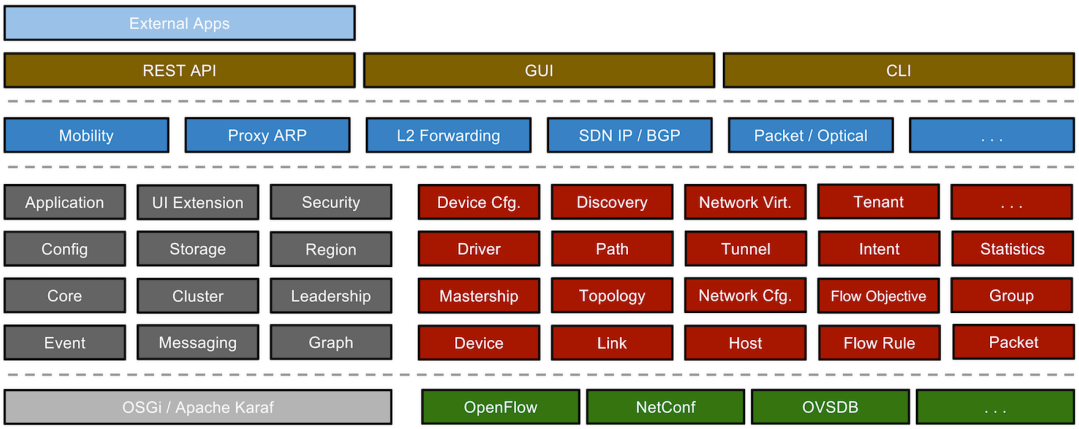


Figura 4.2: Subsistemas de ONOS según [2].

sistemas están formados por componentes que residen en una de las tres capas principales de ONOS (interfaz *Northbound*, *Core* e interfaz *Southbound*), y pueden ser identificados por una o más de las interfaces Java que implementa. La Figura 4.3 resume la relación entre los componentes de los subsistemas. Las líneas de puntos representan los límites entre la interfaz *Northbound*, el *Core* y la interfaz *Southbound*.

4.2.2 Provider

La capa Provider es la más baja de la pila de ONOS, los *Providers* se comunican con la red a través de librerías de protocolos específicas, y con el *Core* a través de la interfaz *ProviderService*.

Los *Protocol-aware Providers* son los responsables de interactuar con el entorno de red utilizando varios protocolos de control y configuración, y proporcionan datos específicos sobre los servicios al *Core*. Algunos *Providers* pueden tener que aceptar peticiones del *Core* y aplicarlas a la red utilizando los protocolos apropiados.

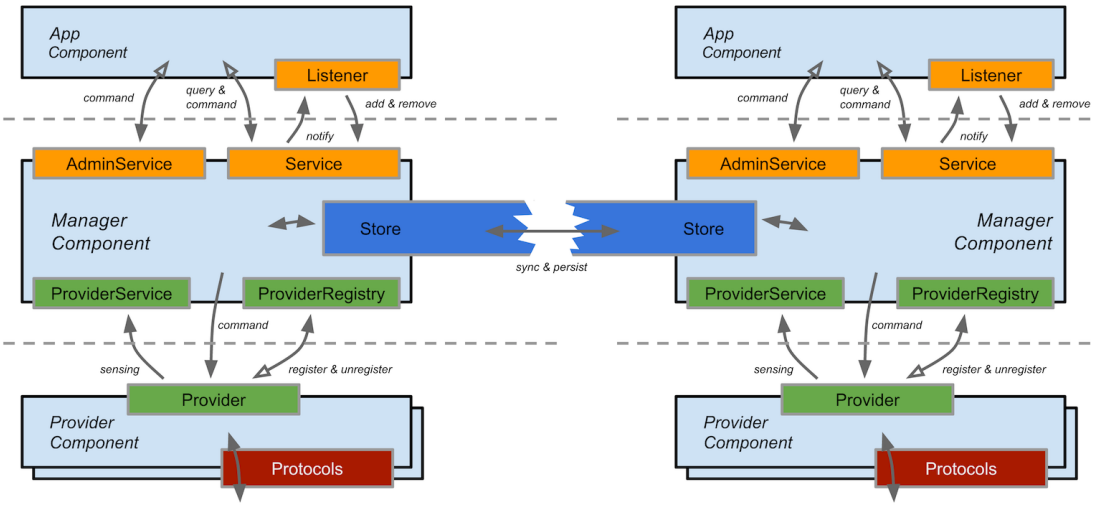


Figura 4.3: Estructura de los subsistemas de ONOS según [2].

Todo *Provider* tiene un identificador (*ProviderId*) asociado. El propósito de esto es proporcionar una identidad externa, lo que permite a los dispositivos y otras entidades permanecer asociadas a la identidad del *Provider* responsable de su existencia incluso después de que el *Provider* haya sido desinstalado o descargado.

4.2.3 Manager

Manager es un componente que reside en el *Core* de ONOS. El *Manager* recibe información de los *Providers* y la proporciona a las aplicaciones. Se compone de las siguientes interfaces:

- *Northbound Service*: interfaz a través de la cual las aplicaciones u otros componentes del *Core* aprenden sobre aspectos concretos del estado de la red.
- *AdminService*: interfaz para lanzar comandos administrativos y aplicarlos al estado de la red en el sistema.
- *Southbound ProviderRegistry*: interfaz a través de la cual los *Providers* pueden registrarse con el *Manager* e interactuar con el.
- *Southbound ProviderService*: interfaz presentada a los *Providers* registrados a través de la que pueden enviar y recibir información a un *Manager*.

4.2.4 Store

Store es un componente que reside en el *Core* de ONOS y está asociado al *Manager*. Los *Stores* indexan, proporcionan persistencia y sincronizan la información recibida desde los *Ma-*

nagers. Esto incluye asegurar la consistencia y robustez de la información a través de las instancias de ONOS.

4.2.5 Aplicaciones

Las aplicaciones consumen y manipulan información agregada por los *Managers* a través de las interfaces *AdminService* y *Service*. Cada aplicación tiene un identificador *(ApplicationId)* único. Este identificador es utilizado por ONOS para controlar el contexto asociado a la aplicación. Para obtener un identificador válido, las aplicaciones deben registrarse contra el *CoreService* proporcionando su nombre (e.g. org.onoslab.onos.fwd).

4.2.6 Descriptions y Events

Las *Descriptions* se utilizan para pasar información sobre un elemento a través del *API Southbound*.

Los *Events* son utilizados por los *Managers* para notificar a los *listeners* cuando ocurre algún cambio en la red. Los *Events* son generados por el *Store* basados en las entradas proporcionadas por el *Manager*. Una vez son generados, se envían a los *listeners* de interés a través de la interfaz *StoreDelegate*. Además, la interfaz *EventDeliveryService* se asegura de que el *Event* solo llegue a los *listeners* deseados. Los *Event listeners* son clases internas de un *Manager* o aplicación e implementan la interfaz *EventListener*. En la Figura 4.4 puede verse la relación entre *Descriptions* y *Events*.

4.3 Estructura de una aplicación ONOS

Las aplicaciones existentes en ONOS siguen la jerarquía de directorios estándar de Maven [20], es decir, el código fuente de aplicaciones y librerías se encuentran bajo *src/main/java* y el código fuente de los *tests* se encuentran bajo *src/test/java*. Esto es debido a que ONOS utilizaba Maven como herramienta para la gestión y construcción de sus aplicaciones. Sin embargo, realizaron una migración a Bazel, y actualmente mantienen dicha jerarquía de directorios pues también es el estándar para Bazel [21].

Además, ONOS añade otros tres directorios a esta jerarquía, quedando el código fuente de la aplicación bajo *src/main/java/org/onosproject/nombreAplicacion* y el código fuente de los *tests* bajo *src/test/java/org/onosproject/nombreAplicacion*.

Fuera de esta jerarquía, al mismo nivel que el directorio *src*, debe existir un fichero BUILD el cual indicará a Bazel qué compilar y cómo hacerlo, especificando los artefactos y sus dependencias.

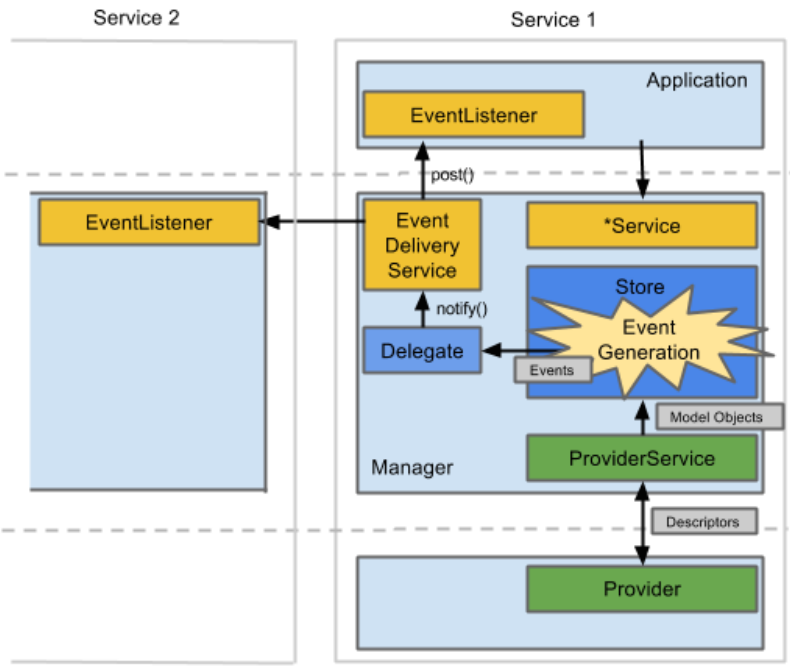


Figura 4.4: Relación entre *Descriptions* y *Events* según [2].

4.3.1 Ficheros código fuente

Toda aplicación ONOS debe implementar un método *Activate* y un método *Deactivate* en al menos uno de sus ficheros de código fuente. En estos métodos, tal y como su nombre indica, se especifica qué debe hacerse al activar la aplicación y qué debe hacerse al desactivarla.

A parte de estos, pueden crearse tantos métodos como sean necesarios para el desarrollo de la aplicación.

4.3.2 Fichero BUILD

Los ficheros BUILD de las aplicaciones ONOS siguen una estructura diferente a la estándar utilizada en Bazel. Normalmente se utilizan las siguientes secciones:

- *COMPILE_DEPS* : en ella se indican las dependencias de compilación de la aplicación.
- *BUNDLES* : en ella se indican los artefactos.
- *osgi_jar* : en ella se indican las dependencias OSGi especificadas en COMPILE_DEPS.
- *onos_app* : en ella se indican la categoría, descripción, los *bundles* incluidos (con la etiqueta *included_bundles*), el título y la url de ONOS.

4.4 Prometheus

Prometheus [3] es un *toolkit open-source* de monitorización y alertas del sistema. Ha sido utilizado en este trabajo como herramienta a la cual exportar las métricas obtenidas en la aplicación desarrollada en ONOS.

El ecosistema de Prometheus consiste en múltiples componentes, siendo muchos de ellos opcionales:

- El servidor principal de Prometheus, el cual recolecta y almacena los datos temporales.
- Una serie de librerías cliente para proporcionar el código de las aplicaciones. Un *gateway* para permitir *jobs* de corta duración.
- *Exporters* de propósito especial para servicios como HAProxy, StatsD, Graphite, etc.
- Un *alertmanager* para controlar las alertas.
- Varias herramientas de soporte.

La mayoría de los componentes de Prometheus están escritos en Go, facilitando su compilado y despliegue como binarios estáticos.

En la Figura 4.5 se puede observar la arquitectura de Prometheus y alguno de sus componentes.

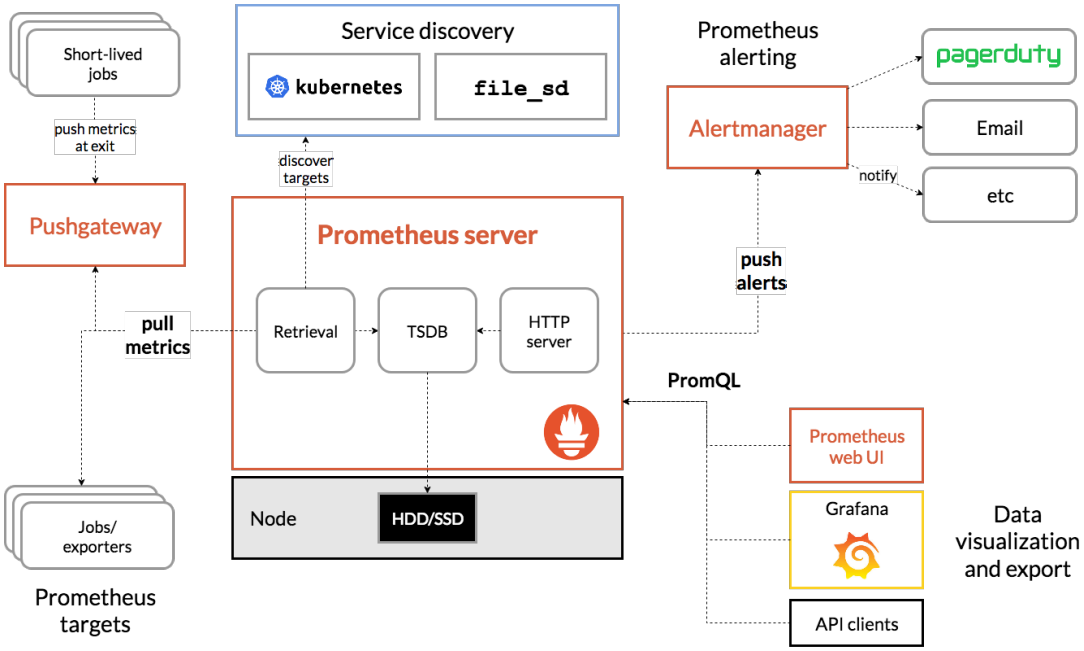


Figura 4.5: Arquitectura de Prometheus según [3].

Prometheus recolecta métricas bien de *jobs* o bien directamente a través de *gateways* configurados para *jobs* de corta duración. Prometheus almacena todos los datos recolectados localmente y corre reglas sobre estos datos para guardarlos como un nuevo conjunto de datos o para generar alertas. Grafana u otras APIs pueden ser utilizadas para visualizar los datos recogidos.

Prometheus funciona bien para recoger cualquier dato temporal puramente numérico. Se ajusta tanto para la monitorización de una máquina como para la monitorización altamente dinámica de arquitecturas orientadas a servicios. En un mundo de microservicios, su soporte para colecciones de datos multi-dimensionales y realización de consultas se convierte en su mayor ventaja.

Prometheus se ha diseñado para proporcionar fiabilidad, siendo el sistema al que se recurre durante un apagón, permitiendo realizar un diagnóstico de los errores y problemas de manera rápida y sencilla. Cada servidor Prometheus es *standalone*, no depende del almacenamiento de la red o de otros servicios remotos. Por lo que se puede depender de Prometheus cuando otras partes de la infraestructura no están funcionando. Siempre es posible visualizar qué estadísticas están disponibles en el sistema, incluso en situaciones de fallo o error del mismo. Si lo que se busca es un nivel de precisión del 100%, como pre-requisito para realizar la facturación, Prometheus no es una buena opción. En este caso se recomienda utilizar otro sistema de recolección y análisis para realizar la facturación y Prometheus para realizar el resto de la monitorización del sistema.

4.5 Desarrollo

En esta sección se detallan los pasos seguidos en el desarrollo de una aplicación ONOS para la exportación de métricas en formato Prometheus.

Para comenzar, se ha creado una aplicación nueva en la carpeta *apps*, existente en el directorio *ONOS_ROOT*, siguiendo la estructura definida en la sección anterior. La jerarquía de directorios y su contenido puede verse en la Figura 4.6 .

```
lara@lara-tfg:~$ cd $ONOS_ROOT
lara@lara-tfg:~/onos$ ls
apps       bazel-onos  bazel-testlogs  cli                 core  Dockerfile  drivers     models     protocols  README.md    tools  web
bazel-bin  bazel-out   BUILD           CODE_OF_CONDUCT.md  deps  docs        LICENSE.txt  pipelines  providers  RELEASES.md  utils  WORKSPACE
lara@lara-tfg:~/onos$ cd apps/prometheuscollector/
lara@lara-tfg:~/onos/apps/prometheuscollector$ ls
BUILD  src
lara@lara-tfg:~/onos/apps/prometheuscollector$ cd src/main/java/org/onosproject/prometheuscollector/
lara@lara-tfg:~/onos/apps/prometheuscollector/src/main/java/org/onosproject/prometheuscollector$ ls
Main.java  package-info.java
```

Figura 4.6: Jerarquía de una aplicación ONOS.

4.5.1 Servicios ONOS

Tras haber creado el fichero en el que residirá el código fuente de la aplicación, se ha creado un esqueleto de la misma. En primer lugar, es necesario utilizar los siguientes servicios ONOS: *Core Service*, *Metrics Service*, *Statistic Service* y *Device Service*. Para ello, debe indicarse de la siguiente forma al principio de la clase principal para cada uno de los servicios deseados. Siendo X el nombre del servicio en cuestión, por ejemplo *core*, *metrics*, etc.:

```
@Reference(cardinality = ReferenceCardinality.MANDATORY)
protected XService XService;
```

Con respecto al funcionamiento de estos servicios, el código que los define reside en el código fuente de ONOS.

Para garantizar el correcto funcionamiento de la apliación desarrollada, ha sido necesario modificar ligeramente el código fuente del servicio *statisticService*. En concreto, se ha modificado la función *highestHitter* existente en el fichero *StatisticManager* añadiendo la siguiente comprobación:

```
if (hitters == null) {
    return null;
}
```

Se ha hecho este cambio ya que de no hacerlo se obtiene un *NullPointerException* al utilizar la función *highestHitter* en la aplicación desarrollada.

4.5.2 Métodos

Continuando el desarrollo de la aplicación, se han creado una serie de métodos en los que se define el comportamiento de la misma. En primer lugar, se ha creado el **método *Activate***. En este se llevan a cabo las siguientes acciones:

- Se registra la aplicación en el servicio *core* de ONOS:

  ```
  appId = coreService.registerApplication("org.onosproject.
      prometheuscollector");
  ```

- Se configuran las *labels* de algunas de las métricas que se desean exportar a Prometheus. Para ello se llama a la función *configLabels*, cuyo código será explicado en profundidad más adelante.

  ```
  SampleBuilder sampleBuilder = configLabels();
  ```

- Se registra el colector de métricas y se exportan las mismas a Prometheus. Dicho colector, *CustomCollector*, se corresponde con una clase privada cuyo código principal será explicado posteriormente.

```
 CustomCollector requests = new CustomCollector().register();
new DropwizardExports(metricsService.getMetricRegistry(),
    sampleBuilder).register();
```

- Se crea, configura y lanza el servidor en el que se expondrán las métricas llamando a la función *serverConfig* cuyo código se explicará en detenimiento más adelante.

En segundo lugar, se ha creado el **método *configLabels***, mencionado previamente, en este se indica cómo configurar las *labels* de diferentes métricas a exportar. Para cada una de las métricas en cuestión, se ha realizado lo siguiente:

- Crear un nuevo *MapperConfig* e indicar el nombre de la métrica para la cual se desean configurar las *labels* y el nombre que se quiere que tenga tras esta configuración. Esto nos permite agrupar bajo una misma métrica, con el mismo nombre, a varias métricas que existían con nombres diferentes. Por ejemplo, si existiese una métrica para cada *switch* de la red llamada *X.ORIGINAL.metrica*, siendo X el identificador del *switch* en cuestión, podrían agruparse bajo una misma métrica *ORIGINAL.metrica* e indicar el identificador del *switch* en una *label*.

```
MapperConfig config = new MapperConfig();
config.setMatch("*.NOMBRE_ORIGINAL.metrica");
config.setName("NOMBRE_MODIFICADO.metrica");
Map<String, String> labels = new HashMap<String, String>();
labels.put("deviceId", "${0}");
config.setLabels(labels);
```

- Una vez hecha la configuración anterior para todas las métricas deseadas, se han añadido todas ellas a una lista de configuraciones. Esta lista de configuraciones se necesita para construir el *sampleBuilder* que devuelve este método, necesario para la exportación de las métricas.

```
List<MapperConfig> list = new ArrayList();
list.add(config);
.
.
.
list.add(configN);
SampleBuilder sampleBuilder = new CustomMappingSampleBuilder(list);
return sampleBuilder;
```

A continuación, se ha creado el **método *serverConfig*** en el que se engloba el código utilizado para la creación, configuración y lanzamiento del servidor necesario para exportar las métricas. Para ello se ha creado un servidor Jetty en el puerto "1234", se ha añadido un *Context Handler*, se ha añadido un *servlet* en la ruta */metrics* y, por último, se ha lanzado el servidor.

```
// Create the server
Server server = new Server(1234);

// Add ServletContextHandler
ServletContextHandler servletContextHandler = new
    ServletContextHandler(ServletContextHandler.SESSIONS);
servletContextHandler.setContextPath("/");
server.setHandler(servletContextHandler);

// Add MetricsServlet
ServletHolder holder = new ServletHolder(new MetricsServlet());
servletContextHandler.addServlet(holder, "/metrics");

// Start the server
server.start();
```

En último lugar, se ha creado una **clase privada *CustomCollector*** que permita recoger métricas ONOS en formato Prometheus. Para ello, se ha hecho que extienda de la clase *Collector* de la libería *client_java* de Prometheus. De esta forma, haciendo *override* del método *collect* de dicha clase, se ha modificado su comportamiento realizando lo siguiente:

- Se han creado dos nuevas métricas, *highest_hitter* y *rate*, indicando su respectiva descripción y *labels*:

```
List<MetricFamilySamples> mfs = new ArrayList<MetricFamilySamples>()
    ;

GaugeMetricFamily rate = new GaugeMetricFamily("rate", "CP rate as
    bytes per second",
Arrays.asList("deviceId", "portNumber"));

GaugeMetricFamily hitter = new GaugeMetricFamily("highest_hitter", "
    CP highest hitter",
Arrays.asList("deviceId", "portNumber"));
```

- A continuación, para conseguir la información necesaria para dichas métricas, ha sido necesario solicitar solicitar la lista de dispositivos de la red:

```
    Iterable<Device> devices = deviceService.getDevices();
```

Para cada uno de los dispositivos de la red se obtiene la lista de puertos y, para cada uno de dichos puertos, se obtiene un par compuesto por el número de puerto y el identificador del dispositivo. Con este par se construye un *ConnectPoint* con el cual se realiza una petición al servicio de estadísticas de ONOS para obtener su *rate*. De esta forma se añade la información obtenida a la métrica *rate*. De igual manera que con el *rate*, se solicita la métrica *highestHitter* al servicio de estadísticas de ONOS utilizando el *ConnectPoint* construido.

```
for (Device d : devices) {
    List<Port> ports = deviceService.getPorts(d.id());
    for (int i = 0; i < ports.size(); i++) {
        ConnectPoint cp = new ConnectPoint(d.id(), ports.get(i).
number());
        String dId = d.id().toString();
        String pNum = ports.get(i).number().toString();
        //Getting rate from StatisticsService
        rate.addMetric(Arrays.asList(dId, pNum),
statisticService.load(cp).rate());
        //Getting highest hitter from StatisticsService
        if (statisticService.highestHitter(cp) == null) {
            hitter.addMetric(Arrays.asList(dId, pNum), 0);
        } else {
            FlowRule fr = statisticService.highestHitter(cp);
            hitter.addMetric(Arrays.asList(dId, pNum), fr.id().
value());
        }
        }
};
```

- Por último, se añaden las métricas creadas a la lista de métricas y se devuelven como salida del método *collect*.

```
mfs.add(hitter);
mfs.add(rate);
return mfs;
```

En el Capítulo C se adjunta el código completo de la aplicación desarrollada.

4.5.3 Aplicaciones ONOS a activar

Para que la aplicación creada funcione correctamente es necesario activar una serie de aplicaciones existentes en el código fuente de ONOS:

- **Reactive Forwarding (fwd)** : proporciona tráfico entre *end-stations* utilizando programación de flujos *hop-by-hop*, interceptando paquetes para los cuales no hay objetivos de flujo en el plano de datos. Las rutas encontrados de esta forma tienen un período de vida corto, i.e. expiran pocos segundos después de que el flujo en el que estaban programadas se detiene. La aplicación relega en el servicio de rutas de ONOS para computar las rutas más cortas. En el caso de eventos de topología negativos (pérdida de enlaces, desconexión de dispositivos, etc.), la aplicación invalidará cualquier ruta cuya programación delegue en algún recurso que ya no esté disponible.

- **Topology & Intent Metrics (metrics)** : aplicación para la monitorización de métricas relacionadas con cambios de topología y programación de *intents*.

- **Control Plane Manager (cpman)** : aplicación para la monitorización de la salud del cluster de ONOS.

- **Packet Statistics (packet-stats)** : aplicación para calcular el número de paquetes de cada tipo.

- **Openflow** : conjunto de aplicaciones englobadas bajo este nombre las cuales activan el protocolo Openflow.

La descripción de dichas aplicaciones ha sido obtenida a partir del fichero BUILD de cada una de ellas. Estos ficheros pueden ser consultados en el código fuente de ONOS en su página de Github. [19]

4.5.4 Dependencias externas

Se ha modificado el fichero de dependencias, *deps.json*, existente en el código fuente de ONOS. Concretamente se han cambiado una serie de dependencias a una versión más reciente y se han añadido otras nuevas. Tras ello es necesario volver a generar las dependencias utilizando el comando *onos-lib-gen*, añadirlas al fichero BUILD de la propia aplicación y volver a compilar tanto ONOS como la aplicación en cuestión. En el Cuadro 4.1 se muestran las dependencias modificadas y sus respectivas versiones.

Librería	Versión
influxdb-java	2.17
simpleclient	0.8.1
simpleclient_common	0.8.1
simpleclient_hotspot	0.8.1
simpleclient_servlet	0.8.1
simpleclient_httpserver	0.8.1
simpleclient_dropwizard	0.8.1
simpleclient_pushgateway	0.8.1
retrofit	2.6.2
converter-moshi	2.6.2
okhttp	3.14.4
logging-interceptor	3.14.4
okio	1.17.2
msgpack-core	0.8.18
jetty-util	9.4.18.v20190429
jetty-websocket-api	9.4.18.v20190429
jetty-websocket	9.4.18.v20190429
jetty-server	9.4.18.v20190429
jetty-servlet	9.4.18.v20190429
jetty-security	9.4.18.v20190429
jetty-continuation	9.4.18.v20190429
jetty-http	9.4.18.v20190429
jetty-io	9.4.18.v20190429

Cuadro 4.1: Librerías utilizadas y sus versiones.

4.6 Compilación y ejecución

En primer lugar ha de compilarse la aplicación, para ello debe lanzarse el siguiente comando en una terminal bajo el directorio ONOS_ROOT. (Figura 4.7)

```
bazel test //apps/prometheuscollector/...
```

Este comando genera un fichero *.oar* que permitirá instalar y activar la aplicación posteriormente.

```
lara@lara-tfg:~$ cd $ONOS_ROOT
lara@lara-tfg:~/onos$ bazel test //apps/prometheuscollector/...
Starting local Bazel server and connecting to it...
INFO: Analyzed 10 targets (148 packages loaded, 2401 targets configured).
INFO: Found 9 targets and 1 test target...
INFO: Deleting stale sandbox base /home/lara/.cache/bazel/_bazel_lara/3627ab32f4e29bdef4b1b317ae17c24
2/sandbox
INFO: Elapsed time: 60.561s, Critical Path: 13.10s
INFO: 0 processes.
INFO: Build completed successfully, 1 total action

INFO: Build completed successfully, 1 total action
```

Figura 4.7: Compilación de la aplicación.

Tras ello, debe abrirse una terminal bajo el directorio ONOS_ROOT y lanzar el comando:

```
bazel run onos-local -- clean debug
```

Esto realizará la ejecución de ONOS, lo cual puede tardar varios minutos.

Una vez ejecutado ONOS ya se puede acceder a su línea de comandos utilizando el siguiente comando en una terminal bajo el directorio ONOS_ROOT. (Figura 4.9).

```
tools/test/bin/onos localhost
```

Para activar las aplicaciones existentes en ONOS, Figura 4.9, ha de accederse a la línea de comandos de ONOS y lanzarse el comando:

```
app activate org.onosproject.nombreAplicacion
```

Para activar la aplicación desarrollada, ha de instalarse previamente lanzando el comando que se muestra a continuación desde la ruta en la que Bazel haya generado el fichero *.oar*. (Figura 4.8).

```
onos-app localhost install onos-apps-nombreAplicacion-oar.oar
```

```
lara@lara-tfg:~$ cd ~/.cache/bazel/_bazel_lara/3627ab32f4e29bdef4b1b317ae17c242/execroot/org_onosproj
ect_onos/bazel-out/k8-fastbuild/bin/apps/prometheuscollector/
lara@lara-tfg:~/.cache/bazel/_bazel_lara/3627ab32f4e29bdef4b1b317ae17c242/execroot/org_onosproject_on
os/bazel-out/k8-fastbuild/bin/apps/prometheuscollector$ onos-app localhost install onos-apps-promethe
uscollector-oar.oar
{"name":"org.onosproject.prometheuscollector","id":193,"version":"2.3.1.SNAPSHOT","category":"Monitor
ing","description":"Monitoring of various metrics related to topology mutation and intent programming
.","readme":"Monitoring of various metrics related to topology mutation and intent programming.","ori
gin":"ONOS Community","url":"http://onosproject.org","featuresRepo":"mvn:org.onosproject/onos-apps-pr
ometheuscollector/2.3.1-SNAPSHOT/xml/features","state":"INSTALLED","features":["onos-apps-prometheusc
ollector"],"permissions":[],"requiredApps":[]}
```

Figura 4.8: Instalación de una aplicación ONOS.

Una vez hecho esto, debe instalarse desde la línea de comandos de ONOS utilizando el comando:

```
feature:install onos-apps-nombreapp
```

Y activarse de igual forma que el resto de aplicaciones con el comando:

```
app activate org.onosproject.nombreAplicacion
```

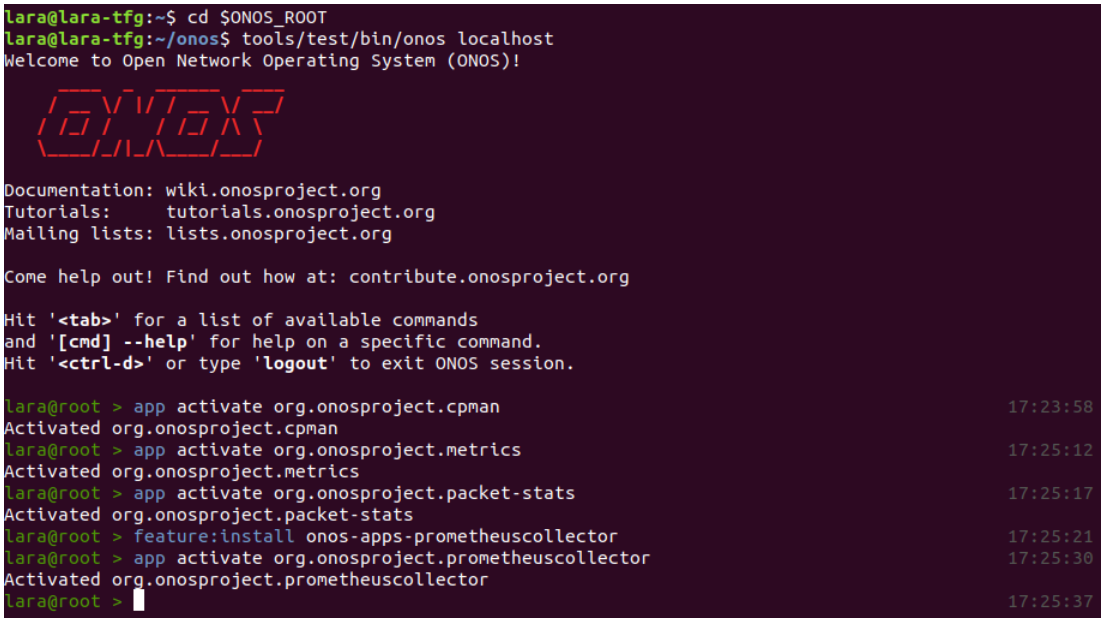
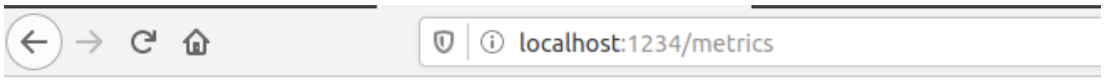


Figura 4.9: Línea de comandos de ONOS.

Después de haber seguido todos los pasos indicados anteriormente, ONOS ya estará corriendo con todas las aplicaciones que hayan sido activadas. Tras ello, ha de lanzarse la topología Mininet explicada en el Capítulo 3. Finalmente, accediendo a través de un navegador a la ruta *localhost:1234/metrics* podrán verse las métricas exportadas. En la Figura 4.10 puede verse una parte de dichas métricas, en concreto *rate* y *highest hitter*.




```
# HELP highest_hitter CP highest hitter
# TYPE highest_hitter gauge
highest_hitter{deviceId="of:0000000000000003",portNumber="LOCAL",} 0.0
highest_hitter{deviceId="of:0000000000000003",portNumber="1",} 0.0
highest_hitter{deviceId="of:0000000000000003",portNumber="2",} 0.0
highest_hitter{deviceId="of:0000000000000003",portNumber="3",} 0.0
highest_hitter{deviceId="of:0000000000000003",portNumber="4",} 0.0
highest_hitter{deviceId="of:0000000000000004",portNumber="LOCAL",} 0.0
highest_hitter{deviceId="of:0000000000000004",portNumber="1",} 0.0
highest_hitter{deviceId="of:0000000000000004",portNumber="2",} 0.0
highest_hitter{deviceId="of:0000000000000004",portNumber="3",} 0.0
highest_hitter{deviceId="of:0000000000000001",portNumber="LOCAL",} 0.0
highest_hitter{deviceId="of:0000000000000001",portNumber="1",} 8.44426036846666E14
highest_hitter{deviceId="of:0000000000000001",portNumber="2",} 0.0
highest_hitter{deviceId="of:0000000000000001",portNumber="3",} 8.44428940346633E14
highest_hitter{deviceId="of:0000000000000002",portNumber="LOCAL",} 0.0
highest_hitter{deviceId="of:0000000000000002",portNumber="1",} 8.44426815122617E14
highest_hitter{deviceId="of:0000000000000002",portNumber="2",} 0.0
highest_hitter{deviceId="of:0000000000000002",portNumber="3",} 0.0
highest_hitter{deviceId="of:0000000000000002",portNumber="4",} 0.0
# HELP rate CP rate as bytes per second
# TYPE rate gauge
rate{deviceId="of:0000000000000003",portNumber="LOCAL",} 0.0
rate{deviceId="of:0000000000000003",portNumber="1",} 0.0
rate{deviceId="of:0000000000000003",portNumber="2",} 0.0
rate{deviceId="of:0000000000000003",portNumber="3",} 0.0
rate{deviceId="of:0000000000000003",portNumber="4",} 0.0
rate{deviceId="of:0000000000000004",portNumber="LOCAL",} 0.0
rate{deviceId="of:0000000000000004",portNumber="1",} 0.0
rate{deviceId="of:0000000000000004",portNumber="2",} 0.0
rate{deviceId="of:0000000000000004",portNumber="3",} 0.0
rate{deviceId="of:0000000000000001",portNumber="LOCAL",} 0.0
rate{deviceId="of:0000000000000001",portNumber="1",} 0.0
rate{deviceId="of:0000000000000001",portNumber="2",} 0.0
rate{deviceId="of:0000000000000001",portNumber="3",} 1.781671419E9
rate{deviceId="of:0000000000000002",portNumber="LOCAL",} 0.0
rate{deviceId="of:0000000000000002",portNumber="1",} 1.728847994E9
rate{deviceId="of:0000000000000002",portNumber="2",} 0.0
rate{deviceId="of:0000000000000002",portNumber="3",} 0.0
rate{deviceId="of:0000000000000002",portNumber="4",} 0.0
```

Figura 4.10: Métricas exportadas.

Capítulo 5

Monitorización de una red SDN

En este capítulo se explican en detalle las herramientas utilizadas para la monitorización de la red definida por software emulada. Cabe recordar que la interacción de dichas herramientas puede verse en la Figura 1.2. Además, se muestran los gráficos y *dashboards* creados.

5.1 Grafana

Grafana [22] es una herramienta *open-source* de visualización y análisis. Permite realizar consultas, crear visualizaciones y alertas y explorar las métricas independientemente de donde estén almacenadas.

De entre sus características cabe destacar la creación de gráficos rápidos y flexibles, la opción de añadir más funcionalidades mediante la instalación de *plugins*, la creación de *dashboards* dinámicos y la posibilidad de utilizar varias fuentes de datos al crear un gráfico.

5.1.1 Fuentes de datos

Grafana permite añadir fuentes de datos de manera sencilla desde su interfaz gráfica. Simplemente se debe acceder a la sección referente a las fuentes de datos existente en el apartado de configuración.

Grafana soporta oficialmente las siguientes fuentes de datos: AWS CloudWatch, Azure Monitor, Elasticsearch, Google Stackdriver, Graphite, InfluxDB, Loki, Microsoft SQL Server, MySQL, OpenTSDB, PostgreSQL, Prometheus y Testdata. Sin embargo, puede añadirse alguna otra fuente de datos utilizando *plugins*.

5.1.2 Visualización

Los elementos más importantes a la hora de visualizar los datos en Grafana son los paneles y los *dashboards*.

Los paneles son el bloque básico de visualización en Grafana. Cada panel tiene un editor de consultas específico para la fuente de datos seleccionada en el mismo. A excepción de algún panel de uso especial, un panel es una representación visual de una o más consultas, las cuales muestran los datos en el tiempo. Estos paneles pueden ser colocados en cualquier lugar dentro de un *dashboard*, además de ser escalables al tamaño deseado por el usuario.

Un *dashboard* es un conjunto de uno o más paneles organizados y colocados en una o más filas. Cada panel puede interactuar con datos de cualquier fuente de datos configurada en Grafana. Las filas constituyen divisores lógicos que pueden agrupar hasta doce paneles de forma autoescable.

Tanto los paneles como los *dashboards* pueden ser compartidos creando *snapshots* que pueden ser publicadas automáticamente a Raintank. Además, los *dashboards* pueden ser exportados e importados como un fichero JSON.

5.1.3 Alertas y notificaciones

Las alertas permiten identificar problemas del sistema poco después de que se produzcan. Se pueden minimizar las interrupciones del sistema identificando rápidamente cambios no intencionados en el mismo. Las alertas en Grafana constan de dos partes:

- Reglas: determinan el momento en el que se lanza una alerta. Las reglas están definidas por una o más condiciones que se evalúan de forma regular por Grafana.
- Canal de notificación: determina cómo se envía cada alerta. Cuando se cumplen las condiciones de una regla, Grafana notifica a los canales configurados para dicha alerta.

Actualmente Grafana solo consta con un panel que soporte alertas (*Graph panel*).

5.2 Pila ELK

La pila ELK es un conjunto de herramientas concebido para facilitar las tareas de ingesta, procesamiento y visualización de datos. Dichas herramientas son las siguientes:

- **Elasticsearch**: constituye un motor de búsqueda y análisis capaz de indexar información fácilmente. Está pensado para almacenar datos de forma escalable.

- **Logstash**: es una herramienta creada para la ingesta y procesamiento de datos.
- **Kibana**: permite visualizar los datos almacenados en índices de Elasticsearch.
- **Beats**: son herramientas que permiten el transporte de los datos desde otras herramientas a Elasticsearch.

En algunos casos en los que no se necesite realizar un procesado de los datos y simplemente se desee almacenar en Elasticsearch los datos existentes en otra herramienta, se puede prescindir del uso de Logstash y utilizar Beats en su lugar. Sin embargo, si lo que se desea es realizar el procesado de datos, ya sea obtenidos de ficheros, de APIs o recogidos de otras herramientas, se necesita el uso de Logstash para realizar dicha tarea. Beats y Logstash también pueden utilizarse conjuntamente. En la Figura 5.1 se muestra la interacción entre las diferentes herramientas que conforman la pila ELK.

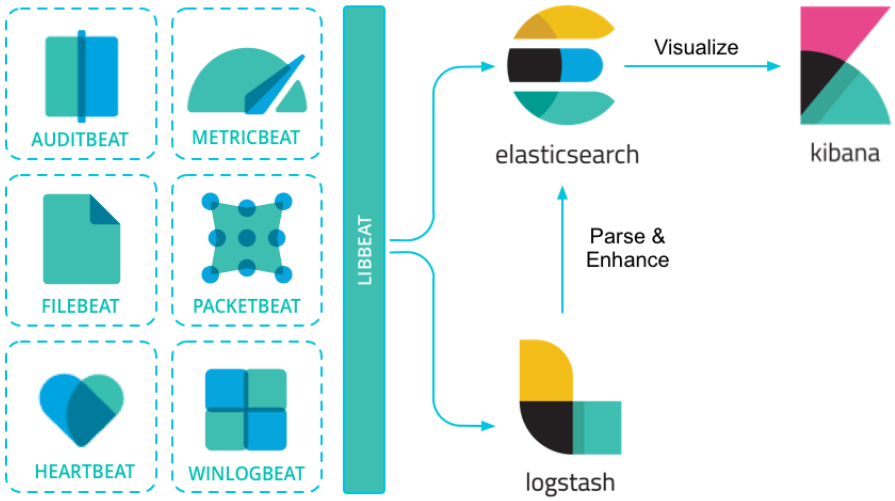


Figura 5.1: Interacción de la pila ELK según [4]

5.2.1 Elasticsearch

Elasticsearch [10] [4] es un motor de búsqueda y análisis, con almacenamiento distribuido y en tiempo real. Puede ser utilizado para muchos propósitos, pero el principal es realizar la indexación de flujos de datos o datos semi-estructurados, tales como *logs* o paquetes de red.

Elasticsearch constituye un almacén de documentos distribuido. En lugar de almacenar la información como filas de una tabla, almacena estructuras complejas que han sido serializadas como documentos JSON. Al tener múltiples nodos en un cluster, los documentos almacenados

están distribuidos a través del cluster y pueden ser accedidos inmediatamente desde cualquier nodo.

Cuando se almacena un documento, se indexa y puede ser buscado rápidamente. Se utiliza una estructura llamada "índice invertido" que soporta búsquedas de texto muy rápidas. Un índice invertido guarda cada una de las palabras que aparecen en cualquier documento e identifican todos los documentos en los que aparece cada palabra.

Se puede pensar en un índice como una colección de documentos optimizada y que cada documento es una colección de campos, que son pares clave-valor que contienen los datos. Por defecto, Elasticsearch indexa los datos en todos sus campos y cada campo indexado tiene una estructura de datos dedicada y optimizada.

Elasticsearch también puede ser *schema-less*, lo que significa que los documentos pueden ser indexados sin especificar explícitamente cómo manejar cada uno de los campos del documento. Cuando se activa el *mapeo dinámico*, Elasticsearch detecta y añade nuevos campos al índice automáticamente. A su vez, se pueden definir reglas para controlar el *mapeo dinámico* y definir mapeos explícitamente para tener un control total sobre como se almacenan e indexan los campos.

Elasticsearch proporciona una forma fácil y rápida de buscar y analizar los datos. En cuanto a la búsqueda de datos, el API REST de Elasticsearch soporta consultas estructuradas, consultas de texto y consultas complejas. Las consultas estructuradas son similares a los tipos de consulta que se pueden realizar en SQL. Las consultas de texto buscan todos los documentos que coincidan con esa cadena de texto y las devolverán ordenadas por relevancia.

En cuanto al análisis de datos, las agregaciones de Elasticsearch permiten crear resúmenes complejos de los datos y ganar visión de las métricas clave, patrones y tendencias. Como las agregaciones tienen las mismas estructuras de datos utilizadas para la búsqueda, también son muy rápidas. Esto permite analizar y visualizar los datos en tiempo real.

5.2.2 Logstash

Logstash [11] [4] es una herramienta *open-source* que permite realizar ingesta de datos y procesarlos en tiempo real. Puede unificar datos de diferentes fuentes dinámicamente y normalizarlos en el destino deseado. Logstash puede recolectar métricas desde muchas infraestructuras y aplicaciones, tales como Ganglia, collectd, NetFlow, JMX y muchos otros, tanto sobre TCP como sobre UDP.

Aunque Logstash fue creado originalmente con el fin de ser una herramienta de recolección de ficheros de *logs*, sus capacidades actuales van más allá. La potencia de Logstash reside en los siguientes aspectos:

- El procesado de datos es escalable horizontalmente gracias a la gran sinergia existente entre Logstash, Elasticsearch y Kibana.
- Su arquitectura, la cual permite mezclar y orquestar diferentes entradas, filtros y salidas.
- Un amplio catálogo de *plugins*, ofreciendo además la posibilidad de contribuir con la creación de nuevos *plugins* a cualquier usuario que lo desee.

Su principal funcionalidad, y para la que esta herramienta se creó, es el manejo y filtrado de ficheros de *logs* y métricas. Logstash puede manejar cualquier tipo de información de *logs*. Puede realizar la ingestión de múltiples *logs* de webs como Apache y aplicaciones como log4j para Java, y, a su vez, capturar muchos formatos de *logs* diferentes, tales como syslog, *logs* sobre las redes, *logs* sobre los *firewalls*, etc.

Otras de sus funcionalidades están ligadas al ámbito web. Logstash permite transformar peticiones HTTP en eventos, pudiendo consumir servicios web (e.g. Twitter) para realizar análisis de seguimiento social, crear alertas y avisos ante diferentes casos de uso, etc. Además, Logstash permite crear eventos mediante sondeos HTTP a *endpoints* bajo demanda.

Logstash también permite el descubrimiento de nuevas formas de emplear los datos. Por ejemplo, entendiendo mejor los datos obtenidos de bases de datos relacionales o NoSQL mediante una interfaz JDBC. O unificando varios *streams* de datos obtenidos de colas de mensajería como Apache Kafka, RabbitMQ y Amazon SQS.

Logstash permite explorar una gran cantidad de datos diferentes, habilitando la captura y captación a través de sensores y la recolección de eventos comunes a través de la ingestión de datos enviados desde dispositivos móviles a casas inteligentes, vehículos, sensores, etc.

El **procesado de eventos** de Logstash se define en sus ficheros de configuración (o *pipelines*) y se compone de tres etapas:

1. *Inputs*: se utilizan para obtener los datos de entrada. Los más utilizados son: *file* que lee los datos desde un fichero existente en el sistema, *syslog* que escucha en el puerto 514 los mensajes syslog y los filtra según el formato RFC3164, *redis* que lee los datos de un servidor redis, y *beats* que procesa eventos enviados por Beats.
2. *Filters*: los filtros realizan el filtrado intermedio en la *pipeline* de Logstash. Se pueden combinar filtros con sentencias condicionales para realizar una acción concreta cuando

un evento coincide con el criterio deseado. Algunos de los filtros más útiles son: *grok* que filtra y estructura texto arbitrario, *mutate* que permite realizar transformaciones en los campos existentes en los eventos, *drop* que permite eliminar un evento por completo, *clone* que permite hacer una copia de un evento, y *geoip* que permite añadir información sobre la localización geográfica de las direcciones IP.

3. *Outputs*: constituyen la última fase del pipeline de Logstash. Un evento puede pasar a través de múltiples salidas, pero una vez que el procesado haya sido completado, el evento finalizará su ejecución. Los *outputs* más utilizados son: *elasticsearch* que permite enviar datos de los eventos a Elasticsearch, *file* que permite escribir datos de los eventos en un fichero del sistema, *graphite* que envía datos de los eventos a Graphite, y *statsd* que envía datos de los eventos a statsd.

Tanto *inputs* como *outputs* soportan *codecs* que permiten codificar y decodificar los datos al entrar o salir de un *pipeline* sin tener que utilizar un filtro por separado. Los *codecs* son filtros que se pueden aplicar a los flujos de datos. Permiten separar el transporte de los mensajes del proceso de serialización de manera sencilla. Los *codecs* más populares son: json, msgpack, y plain (texto plano).

5.2.3 Kibana

Kibana [13] [4] es una herramienta *open-source* que pertenece a Elastic y permite visualizar y explorar los datos indexados en Elasticsearch. Además, Kibana constituye la herramienta de monitorización de la pila ELK, pudiendo tener control del comportamiento y estado de las herramientas que la componen.

Kibana utiliza patrones (*index patterns*) para saber qué índices de Elasticsearch debe explorar. Un *index pattern* puede coincidir con un único índice de Elasticsearch, o incluír un comodín (*) para que coincida con varios índices.

Para crear *dashboards* y visualizaciones Kibana ofrece una serie de funcionalidades, correspondientes a sus diferentes secciones, que facilitan esta labor:

- *Discover*: facilita la exploración de los datos. Entre sus características cabe destacar que permite:
  - Acceder a todos los documentos de cada uno de los índices que coincidan con el *index pattern* seleccionado.
  - Buscar datos y filtrar los resultados de la búsqueda.
  - Obtener detalles a nivel de campo sobre los documentos que coinciden con la búsqueda.

  - Visualizar los eventos que han ocurrido justo antes y después de un documento.

- *Visualize*: permite crear visualizaciones sobre los datos almacenados en los índices de Elasticsearch. Las visualizaciones creadas pueden agruparse para crear *dashboards*. Kibana soporta varios tipos de visualizaciones de los que cabe destacar:

  - *Lens*: permite construir rápidamente varias visualizaciones básicas simplemente arrastrando los campos de los datos que se desea visualizar.
  - *Gráficos de uso común* : gráfico de líneas área o barras, gráfico de tarta, tabla, métrica, *gauge*, etc.
  - *TSVB*: permite visualizar datos temporales utilizando agregaciones de *pipelines*.
  - *Timelion*: computa y combina datos de varios *data sets*.
  - *Elastic Maps*: permite visualizar datos geo-espaciales.

- *Dashboard*: es un conjunto de visualizaciones, búsquedas y mapas, normalmente en tiempo real. Los *dashboards* permiten visualizar los datos desde diferentes perspectivas. Un *dashboard* permite añadir visualizaciones, búsquedas y mapas, organizar los diferentes elementos para mostrar los datos exactamente de la manera deseada, personalizar los rangos temporales para mostrar exactamente el rango deseado e inspeccionar y editar sus elementos para encontrar qué tipo de datos está siendo visualizado.

- *Canvas*: es una herramienta de visualización y presentación de datos que trabaja con Kibana. Permite solicitar datos en vivo directamente a Elasticsearch, y combina los datos por colores, imágenes, texto, etc. De entre las funcionalidades aportadas por esta herramienta cabe destacar que permite crear y personalizar el espacio de trabajo con diferentes fondos, bordes, colores, fuentes, etc., personalizar el panel de trabajo con visualizaciones propias, tales como imágenes o texto, mostrar los datos a través de gráficos, grafos, barras de progreso, etc, personalizar los datos solicitándolos directamente a Elasticsearch, y hacer énfasis en los datos deseados mediante el uso de filtros.

- *Metrics*: es una aplicación que permite monitorizar la infraestructura de las métricas e identificar problemas en tiempo real. Proporciona un resumen de la infraestructura de los datos, a partir de donde se puede profundizar y visualizar métricas más detalladas u otra información.

- *Logs*: es una aplicación que permite explorar los *logs*. En ella se puede filtrar los *logs* por varios campos, ya sea en tiempo real o seleccionando el rango temporal deseado.

Finalmente, además de las funcionalidades nativas en Kibana, se pueden instalar una serie de *plugins* para añadir nuevas funcionalidades a la herramienta. En el caso de querer realizar la

instalación de uno de dichos *plugins* se debe tener en cuenta su compatibilidad con la versión de Kibana que se está utilizando.

5.2.4 Beats

Los Beats son herramientas *open-source* que se instalan como agentes en los servidores para enviar datos operacionales a Elasticsearch.

Elastic proporciona Beats para capturar: datos de auditoría (Auditbeat), ficheros de *log* (Filebeat), datos *cloud* (Functionbeat), disponibilidad (Heartbeat), systemd journals (Journalbeat), métricas (Metricbeat), tráfico de red (Packetbeat) y ventanas de eventos de *logs* (Winlogbeat).

Los Beats pueden enviar los datos directamente a Elasticsearch o a través de Logstash, donde además se pueden procesar los datos antes de visualizarlos en Kibana.

Aunque Elastic solo proporciona los Beats mencionados, permite que los usuarios desarrollen nuevos Beats que se ajusten a sus necesidades. Por ello, proporciona una librería (libbeat) que ofrece el API utilizada por todos los Beats para enviar los datos a Elasticsearch, configurar las opciones de entrada, etc.

5.3 Configuración de pipelines en Logstash

Como se ha mencionado anteriormente, en este trabajo se utiliza Logstash para realizar el filtrado del fichero de *logs* de ONOS, así como el sondeo y procesado de los datos de su API REST. Para poder realizar dichas tareas, se han creado dos ficheros de configuración o *pipelines* bajo la ruta */etc/logstash/conf.d*:

1. Un fichero para filtrar los ***logs* de ONOS**. En primer lugar, se indica la ruta al fichero de *logs* de ONOS como entrada y se utiliza el *codec multiline* de Logstash ya que existen ciertos mensajes de *log* en ONOS que están formados por varias líneas.

```
input {
  file {
    path => "/tmp/onos-2.3.1-SNAPSHOT/onos.log"
    type => "onoslogs"
    codec => multiline {
      pattern => "^\s"
      what => "previous"
    }
  }
}
```

En segundo lugar, se han definido los filtros a aplicar sobre el fichero de *logs*. El filtro *mutate* se ha utilizado para eliminar los códigos de color, los cuales no aportaban información adicional y dificultaban la lectura de los datos. El filtro *grok* se ha utilizado para dividir cada uno de los mensajes de *log* en diferentes partes tales como el *timestamp*, el *loglevel*, etc.

```
filter {
  mutate {
    gsub => ["message", "\x1B\[([0-9]{1,2}(;[0-9]{1,2})?)?[m|K]",
    ""]
  }
  grok {
    match => { "message" => "%{TIME:onostimelog}%{SPACE}%{LOGLEVEL:
    onosloglevel}%{SPACE}[\[]%{DATA:onosclass}[\]]%{SPACE}%{
    GREEDYDATA:onosmessage}" }
  }
}
```

Por último, se indica la salida deseada, en este caso Elasticsearch, creando un nuevo índice "onoslogs" para cada día.

```
output {
  if [type] == "onoslogs" {
    elasticsearch {
      hosts => ["localhost:9200"]
      index => "onoslogs-%{+YYYY.MM.dd}"
    }
  }
}
```

2. Otro fichero para realizar un **sondeo al API REST de ONOS** y procesar la información obtenida. En primer lugar, se utiliza la opción *http_poller* como entrada. Para cada una de las *urls* que se desee sondear, se indicará la *url*, el método HTTP, el usuario, contraseña y cabeceras de la petición. A continuación, se muestra un ejemplo de ello, correspondiente al sondeo de la url de las aplicaciones de ONOS. Una vez realizado esto para todas las *urls* deseadas, se indicará el *timeout* para las peticiones, el período de tiempo cada el cual debe lanzarse una nueva petición y el *codec* deseado, en este caso *json*.

```
input {
   http_poller {
     urls => {
       onos_apps => {
         # Supports all options supported by ruby's Manticore HTTP
    client
```

```
        method => get
        user => "onos"
        password => "rocks"
        url => "http://localhost:8181/onos/v1/applications"
        headers => {
          Accept => "application/json"
        }
      }
      .
      .
      .
    }
    request_timeout => 60
    # Supports "cron", "every", "at" and "in" schedules by rufus
   scheduler
    schedule => { every => "5s"}
    codec => "json"
    # A hash of request metadata info (timing, response headers,
   etc.) will be sent here
    metadata_target => "http_poller_metadata"
    type => "onosapi"
  }
}
```

A continuación, se indican los filtros a aplicar sobre los datos sondeados. En este caso se utiliza el filtro *split* para separar los diferentes campos de los ficheros JSON obtenidos, en campos a crear en el índice de Elasticsearch. En el código expuesto a continuación, se muestran ejemplos para los campos referentes a las aplicaciones, los dispositivos y los flujos de ONOS.

```
filter {
  split {
    field => "applications"
  }
  split {
    field => "devices"
  }
  split {
    field => "flows"
  }
  split {
    field => "[flows][selector][criteria]"
  }
  split {
    field => "[flows][treatment][instructions]"
  }
```

```
  .
  .
  .
}
```

Por último, se indica la salida deseada, en este caso Elasticsearch, creando un nuevo índice "onosapi" para cada día.

```
output {
  if [type] == "onosapi" {
    elasticsearch {
      hosts => ["localhost:9200"]
      index => "onosapi-%{+YYYY.MM.dd}"
    }
  }
}
```

En el Capítulo D se adjunta el código completo de estos ficheros.

5.4 Creación de un *dashboard* utilizando Grafana

En primer lugar, se optó por utilizar Grafana como herramienta para la monitorización de la red debido a su sencillez y fácil integración con Prometheus.

5.4.1 Grafana y Prometheus

Como primera aproximación, se creó un *dashboard* simple mostrando una serie de gráficos sencillos mediante los cuales se pudiese comprobar el correcto funcionamiento de la aplicación. A continuación se explica en detalle cada uno de los gráficos añadidos en dicho *dashboard*. Las imágenes que se muestran han sido generadas por la herramienta de monitorización Grafana durante una ejecución del *script* de Mininet, generando tráfico entre todos los *hosts*, tal y como se ha descrito en el Capítulo 3, durante aproximadamente veinte minutos.

Gráfico *Tipos de eventos de la red* (Figura 5.2): gráfico de tipo tarta que muestra la cantidad de eventos de red de cada tipo. Se optó por este tipo de gráfico ya que además de mostrar el número concreto de eventos, se puede ver con facilidad la cantidad de eventos de un tipo en relación con los demás.

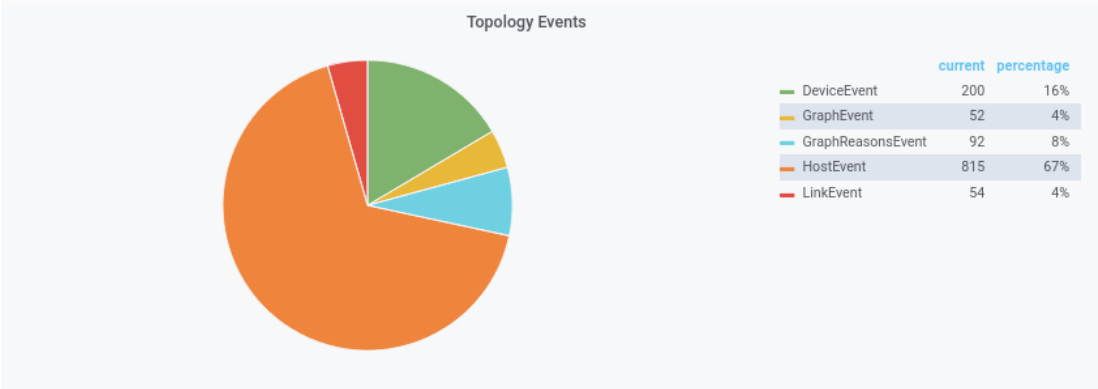


Figura 5.2: Grafana: Tipos de eventos de la red.

Gráfico *Contadores de paquetes* (Figura 5.3): gráfico de barras horizontales en el que se muestra el número de paquetes de cada tipo que pasan a través del controlador. Se ha optado por este tipo de gráfica ya que puede verse el número de paquetes de cada tipo y su relación con los demás.

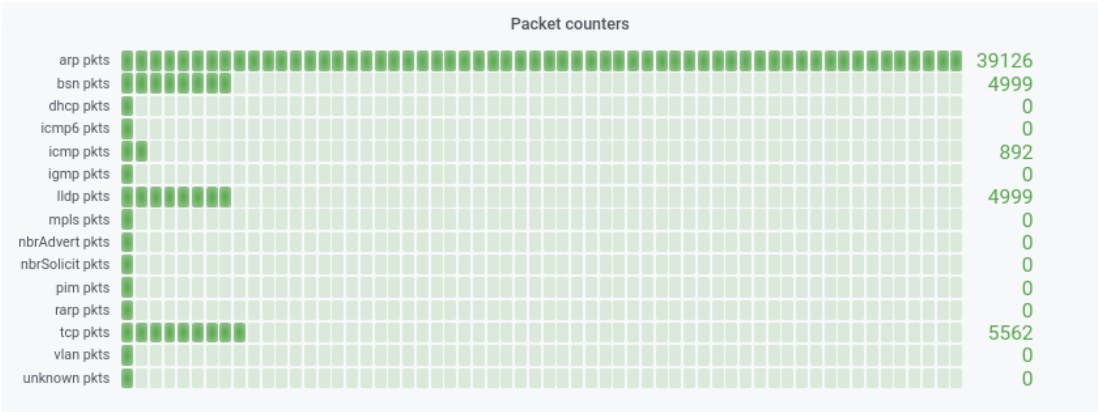


Figura 5.3: Grafana: Contadores de paquetes.

Gráfico *Highest hitter* (Figura 5.4): gráfico temporal de líneas en el que se muestra la métrica calculada *highest hitter*. Al tratarse de un gráfico interactivo, colocando el cursor sobre cada una de las líneas en el tiempo puede observarse el valor en ese momento.

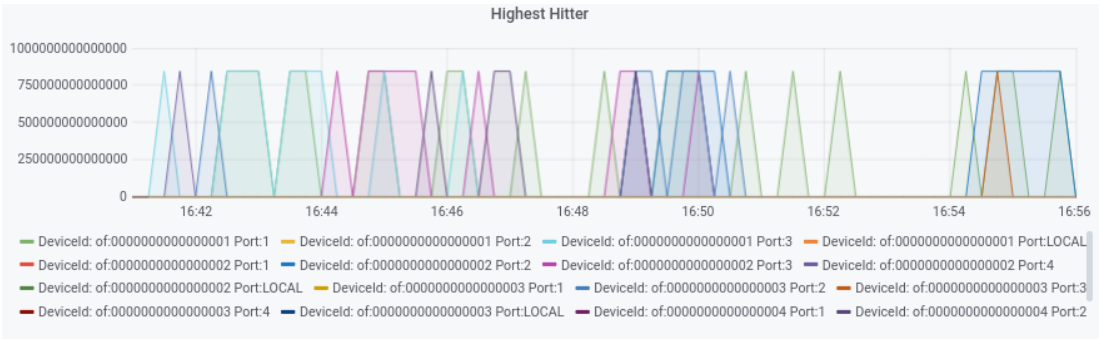


Figura 5.4: Grafana: Highest hitter.

Gráfico *Rate* (Figura 5.5): gráfico temporal de líneas en el que se muestra la métrica calculada *rate*. Al tratarse de un gráfico interactivo, colocando el cursor sobre cada una de las líneas en el tiempo puede observarse el valor del *rate* en ese momento para el puerto y *switch* asociado a dicha línea.

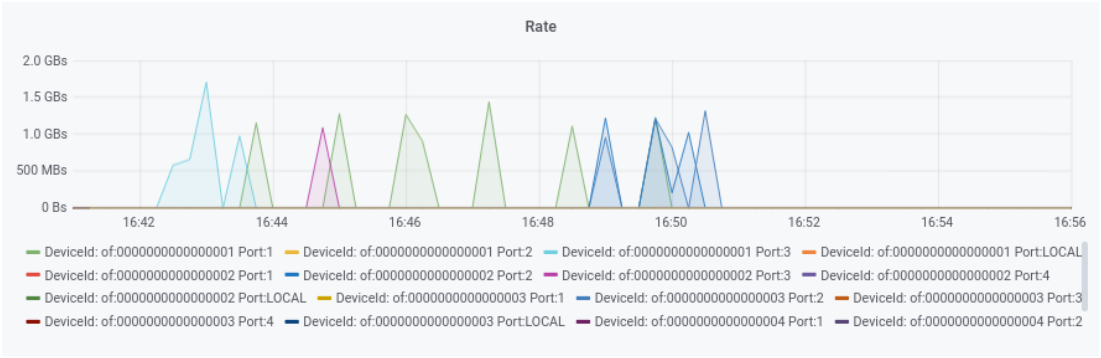


Figura 5.5: Grafana: Rate.

Gráfico *Tráfico de entrada* (Figura 5.6): gráfico de tipo *gauge* en el que se muestra el número de paquetes de entrada en el controlador ONOS para cada uno de los *switches* de la red.

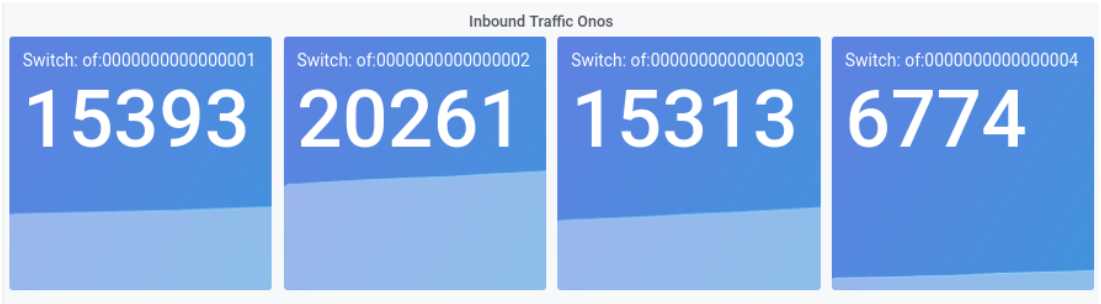


Figura 5.6: Grafana: Tráfico de entrada.

Gráfico *Tráfico de salida* (Figura 5.7): gráfico de tipo *gauge* en el que se muestra el número

de paquetes de salida del controlador ONOS para cada uno de los *switches* de la red.

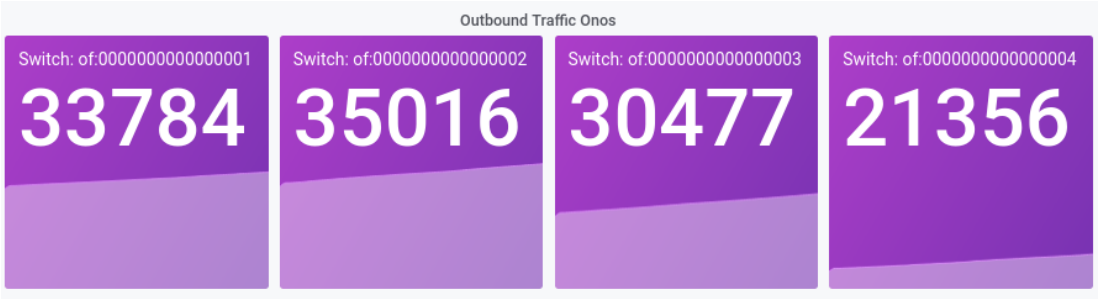


Figura 5.7: Grafana: Tráfico de salida.

Gráfico *Paquetes de entrada por segundo* (Figura 5.8): gráfico temporal de líneas en el que se muestra el número de paquetes de entrada por segundo para cada *switch*. Al tratarse de un gráfico interactivo, desplazando el cursor sobre las diferentes líneas del gráfico se muestra el número de paquetes por segundo en ese momento para el *switch* asociado a dicha línea.

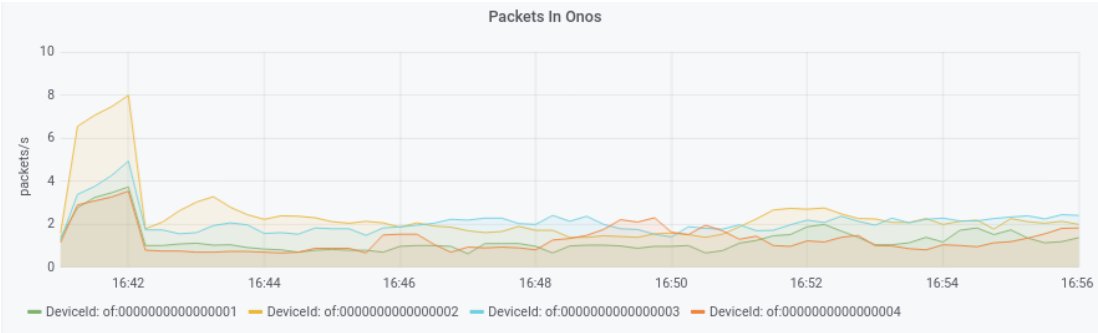


Figura 5.8: Grafana: Paquetes de entrada por segundo.

Gráfico *Paquetes de salida por segundo* (Figura 5.9): gráfico temporal de líneas en el que se muestra el número de paquetes de salida por segundo para cada *switch*. Al tratarse de un gráfico interactivo, desplazando el cursor sobre las diferentes líneas del gráfico se muestra el número de paquetes por segundo en ese momento para el *switch* asociado a dicha línea.

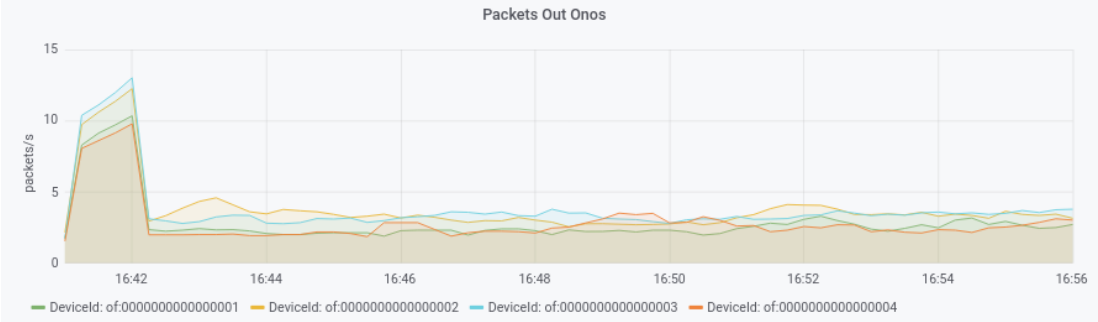


Figura 5.9: Grafana: Paquetes de salida por segundo.

Gráfico *Flujos modificados por segundo* (Figura 5.10): gráfico temporal de líneas en el que se muestra el número de flujos modificados por segundo para cada *switch*. Al tratarse de un gráfico interactivo, desplazando el cursor sobre las diferentes líneas del gráfico se muestra el número de flujos modificados por segundo en ese momento para el *switch* asociado a dicha línea.

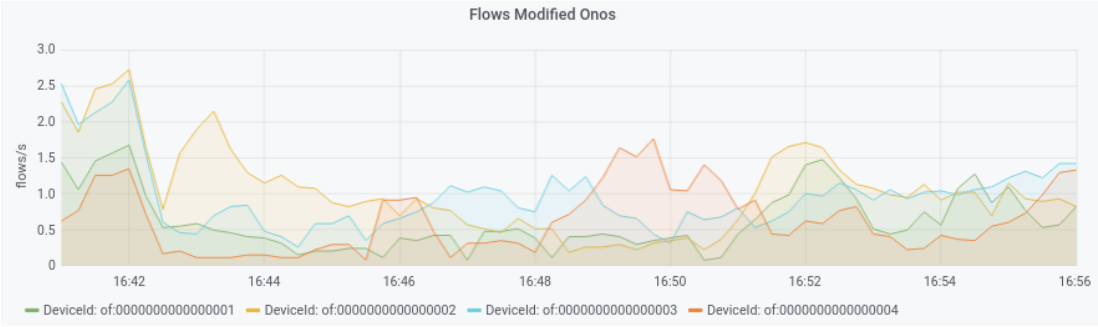


Figura 5.10: Grafana: Flujos modificados por segundo.

Gráfico *Flujos eliminados por segundo* (Figura 5.11): gráfico temporal de líneas en el que se muestra el número de flujos eliminados por segundo para cada *switch*. Al tratarse de un gráfico interactivo, desplazando el cursor sobre las diferentes líneas del gráfico se muestra el número de flujos modificados por segundo en ese momento para el *switch* asociado a dicha línea. En el caso de esta simulación no se ha eliminado ningún flujo.

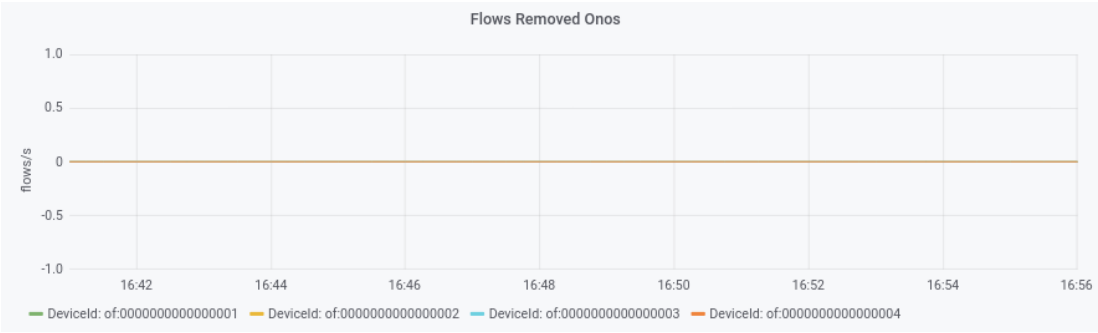


Figura 5.11: Grafana: Flujos eliminados por segundo.

Gráfico *Stats respondidos por segundo* (Figura 5.12): gráfico temporal de líneas en el que se muestra el número de *stats* respondidos por segundo para cada *switch*. Al tratarse de un gráfico interactivo, desplazando el cursor sobre las diferentes líneas del gráfico se muestra el número de *stats* respondidos por segundo en ese momento para el *switch* asociado a dicha línea.

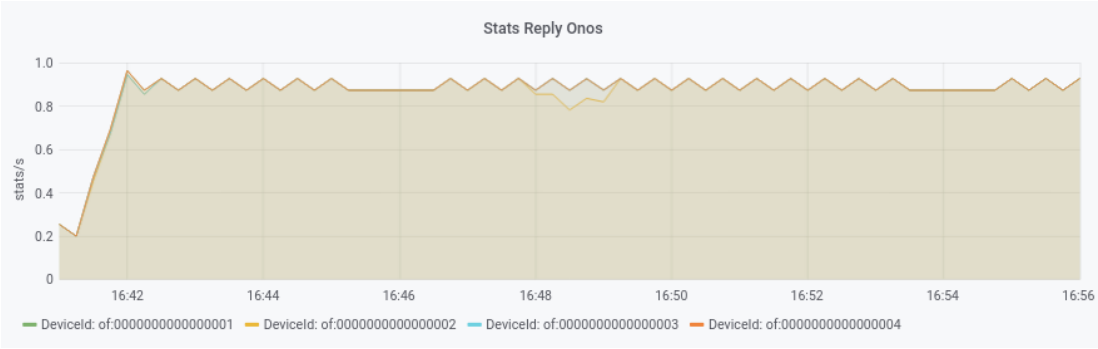


Figura 5.12: Grafana: *Stats* respondidos por segundo.

Gráfico *Stats solicitados por segundo* (Figura 5.13): gráfico temporal de líneas en el que se muestre el número de *stats* solicitados por segundo para cada *switch*. Al tratarse de un gráfico interactivo, desplazando el cursor sobre las diferentes líneas del gráfico se muestra el número de *stats* solicitados por segundo en ese momento para el *switch* asociado a dicha línea.

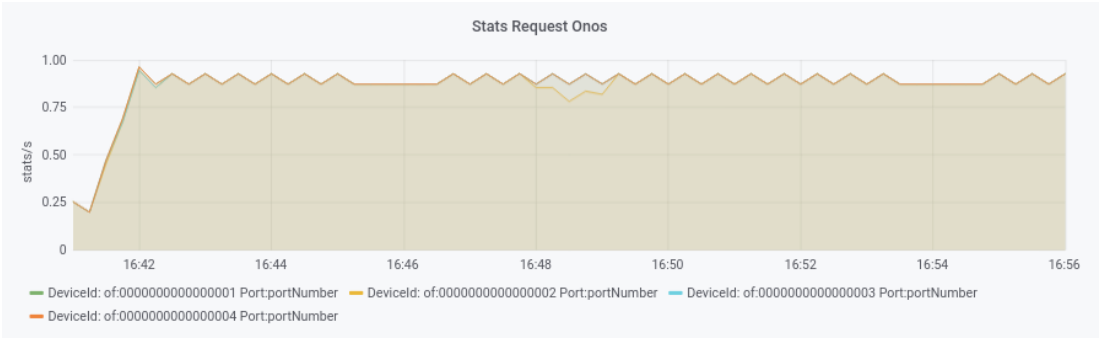


Figura 5.13: Grafana: *Stats* solicitados por segundo.

5.4.2 Grafana, Prometheus y Elasticsearch

Al utilizar únicamente Prometheus para el almacenamiento de métricas, estas solo permanecían almacenadas durante un período corto de tiempo. Para almacenar dichas métricas de manera persistente, se ha utilizado Elasticsearch. Además, se ha empleado Logstash para obtener más información sobre ONOS y la red emulada, creando de este modo dos ficheros en su configuración en los cuales se filtran los *logs* de ONOS y sr procesa y sondea su *API REST*.

Con la nueva información obtenida, se han añadido gráficas al *dashboard* descrito en el apartado anterior. A continuación se muestran imágenes del contenido añadido, al igual que anteriormente, han sido generadas utilizando la herramienta de monitorización Grafana durante una ejecución del *script* de Mininet durante aproximadamente veinte minutos.

Tabla *Processors* (Figura 5.14): tabla en la que se muestran los *processors* de ONOS, su tipo, prioridad y nombre de la clase ONOS.

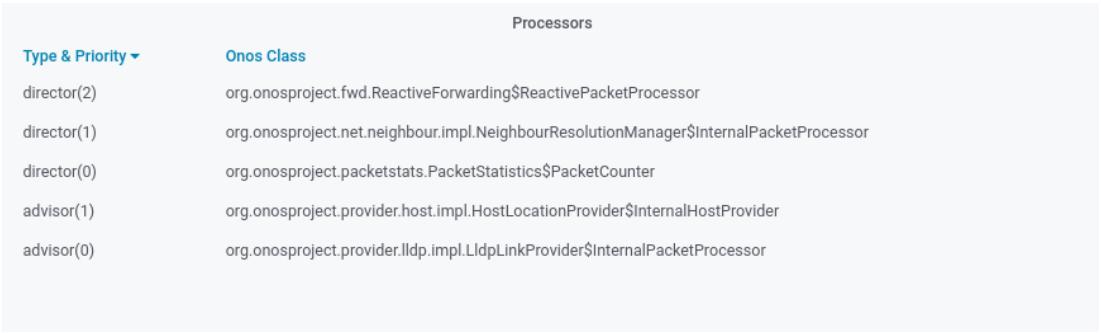


Figura 5.14: Grafana: Processors.

Tabla *Aplicaciones activas* (Figura 5.15): tabla en la que se muestran las aplicaciones activas durante la ejecución de ONOS en esa simulación. Sobre ellas se muestra su nombre, categoría, estado, descripción y versión.

Onos Active Applications

Name	Category	State	Description	Version
org.onosproject.proxyarp	Traffic Engineering	ACTIVE	Proxy ARP\/NDP application.	2.3.1.SNAPSHOT
org.onosproject.packet-stats	Utility	ACTIVE	Application to calculate the number of packets of different types	2.3.1.SNAPSHOT
org.onosproject.optical-model	Optical	ACTIVE	ONOS optical information model.	2.3.1.SNAPSHOT
org.onosproject.openflow-message	Provider	ACTIVE	ONOS OpenFlow control message provider.	2.3.1.SNAPSHOT

Figura 5.15: Grafana: Aplicaciones activas.

Tabla *Hosts* (Figura 5.16): tabla en la que se muestran los *hosts* de la red. Sobre ellos se muestra: dirección IP, identificador, *switch* al que está directamente conectado, número de puerto del *switch* al que está conectado y dirección MAC.

Hosts

Address	Host Id	Switch	Port Number	MAC Address
10.0.0.8	FE:2E:D5:27:36:8D/None	of:0000000000000004	3	FE:2E:D5:27:36:8D
10.0.0.7	9E:C0:7A:01:D1:D8/None	of:0000000000000004	2	9E:C0:7A:01:D1:D8
10.0.0.6	82:8A:98:61:50:5F/None	of:0000000000000003	4	82:8A:98:61:50:5F
10.0.0.5	2A:B2:67:42:43:1C/None	of:0000000000000003	3	2A:B2:67:42:43:1C
10.0.0.4	82:F7:E0:5B:88:4A/None	of:0000000000000002	4	82:F7:E0:5B:88:4A
10.0.0.3	6E:8E:E5:03:31:ED/None	of:0000000000000002	3	6E:8E:E5:03:31:ED
10.0.0.2	0A:8B:CA:38:67:27/None	of:0000000000000001	3	0A:8B:CA:38:67:27

Figura 5.16: Grafana: Hosts.

5.5 Creación de un *dashboard* utilizando ELK

Tal y como se ha indicado previamente en este trabajo, se ha utilizado Elasticsearch para asegurar la persistencia de las métricas, y Logstash para obtener más información acerca de la red y de ONOS. Estas herramientas conforman la E y L de la pila ELK, siendo Kibana la K y la única que no estaba siendo utilizada. Por ello, se optó por utilizarla como herramienta de monitorización, usando así todas las herramientas que forman la pila ELK. Se ha tomado esta decisión ya que, a pesar de que Grafana permite tener a Elasticsearch como fuente de datos, no aporta una forma sencilla mediante la cual recoger y visualizar los *logs* de ONOS. Por ello, utilizar Kibana como herramienta de monitorización implica mostrar la información de una manera más clara y aprovechar las ventajas de integración de la pila ELK. Además, constituye una herramienta de monitorización y visualización muy potente la cual nos permite tener un mayor nivel de precisión en los gráficos.

En primer lugar se creó un *dashboard* en el que se muestra información general de la red, similar al hecho previamente en Grafana. Dicho *dashboard* se compone de los siguientes gráficos:

Gráfico *Switches* (Figura 5.17): gráfico de tipo *Top N* en el que se muestran los *switches* existentes en la red. Al tratarse de un gráfico interactivo, al hacer click sobre cada una de las barras, correspondientes a un *switch*, se redirigirá a un *dashboard* en detalle sobre dicho *switch*.

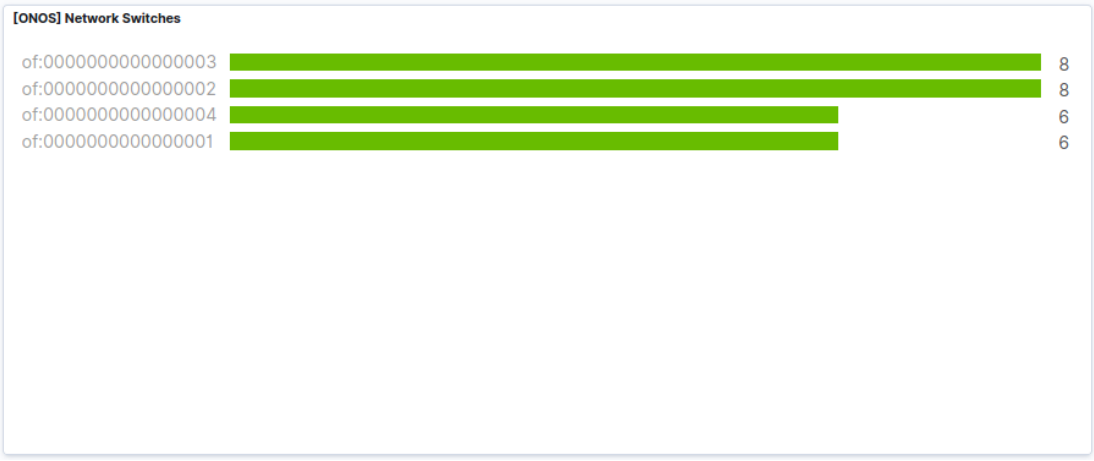


Figura 5.17: Kibana: Switches.

Gráficos *Tráfico de la red* (Figura 5.18): un conjunto de gráficos de tipo *gauge* en los que se muestra el tráfico de entrada y salida de la red tanto en paquetes/s como en bytes/s, además del total de paquetes y bytes.

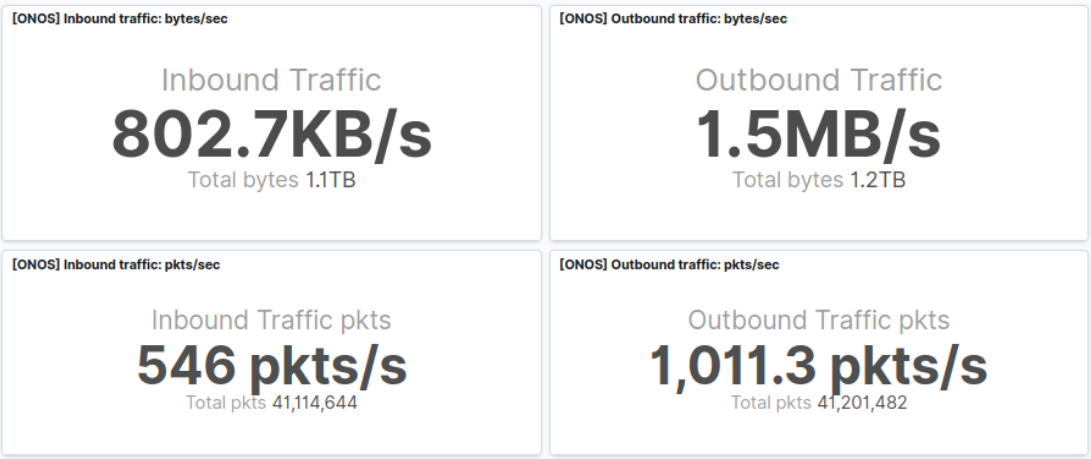


Figura 5.18: Kibana: Tráfico de la red.

Gráfico *Tipos de paquetes* (Figura 5.19): gráfico de tipo *Top N* en el que se muestra la cantidad existente de cada tipo de paquete de la red. Se ha optado por utilizar este tipo de gráfico dado que puede verse tanto el número de cada tipo de paquete como la relación de la cantidad de cada uno con los demás tipos.

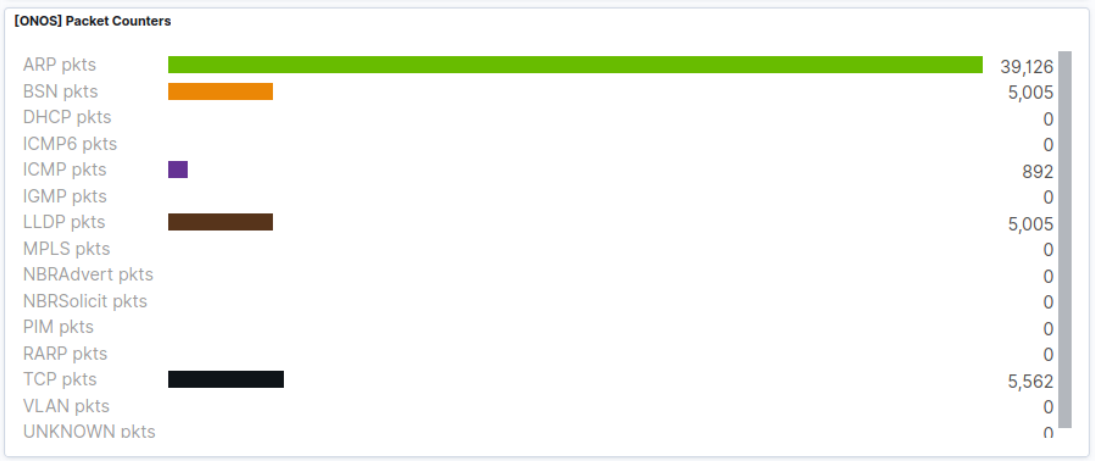


Figura 5.19: Kibana: Tipos de paquetes.

Gráfico *Tipos de eventos* (Figura 5.20): gráfico de tipo *Top N* en el que se muestra la cantidad de cada tipo de evento existente en la red. Se ha optado por utilizar este tipo de gráfico dado que puede verse tanto el número de cada tipo de evento como la relación de este con los demás tipos.

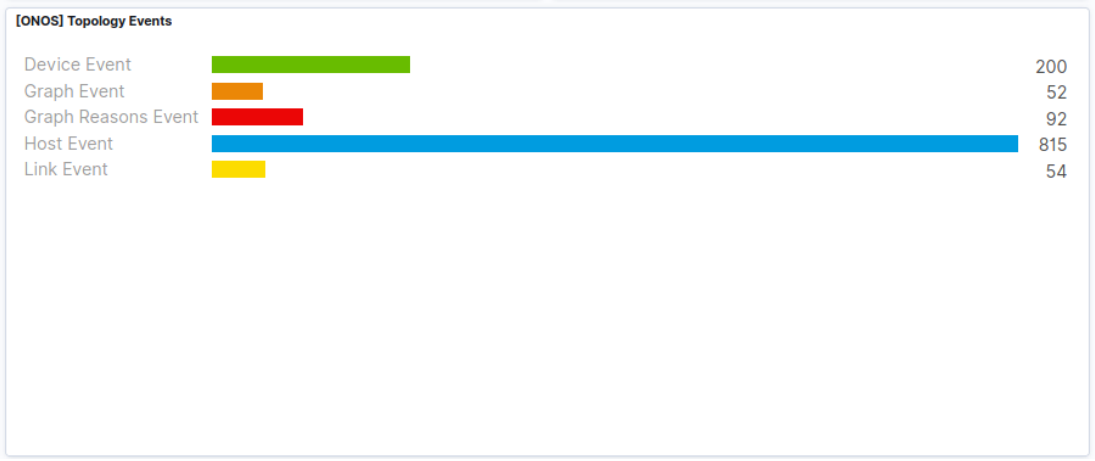


Figura 5.20: Kibana: Tipos de eventos.

Gráfico *Paquetes de entrada* (Figura 5.21): gráfico temporal de líneas en el que se muestran los paquetes de entrada por segundo al controlador ONOS. Al tratarse de un gráfico interactivo, desplazando el cursor sobre las diferentes líneas se mostrará el número de paquetes de

entrada por segundo en ese momento para el *switch* asociado a dicha línea. Se ha observado que Kibana corta el identificador del *switch* en formato OpenFlow. Esperamos que Kibana corrija esto en futuras versiones, de no ser así, habrá que optar por utilizar otro formato de identificador para los *switches*.

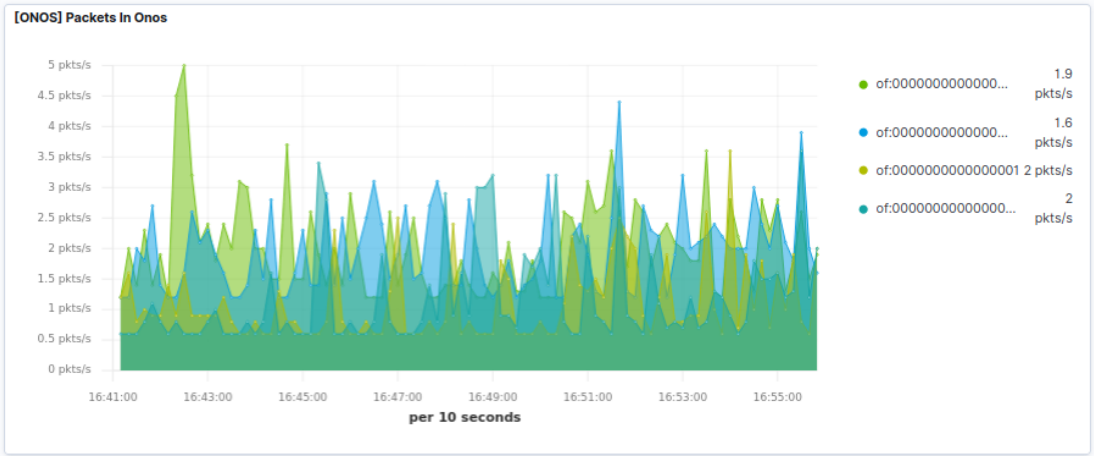


Figura 5.21: Kibana: Paquetes de entrada.

Gráfico *Paquetes de salida* (Figura 5.22): gráfico temporal de líneas en el que se muestran los paquetes de salida por segundo al controlador ONOS. Al tratarse de un gráfico interactivo, desplazando el cursor sobre las diferentes líneas se mostrará el número de paquetes de salida por segundo en ese momento para el *switch* asociado a dicha línea.

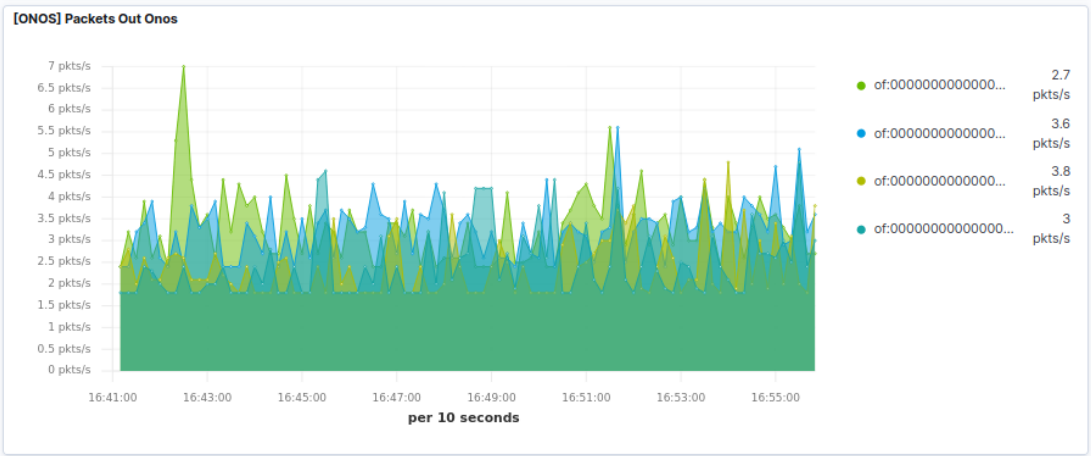


Figura 5.22: Kibana: Paquetes de salida.

Gráfico *Flujos modificados* (Figura 5.23): gráfico temporal de líneas en el que se muestran los flujos modificados por segundo. Al tratarse de un gráfico interactivo, desplazando el cursor

sobre las diferentes líneas se mostrará el número de flujos modificados por segundo en ese momento para el *switch* asociado a dicha línea.

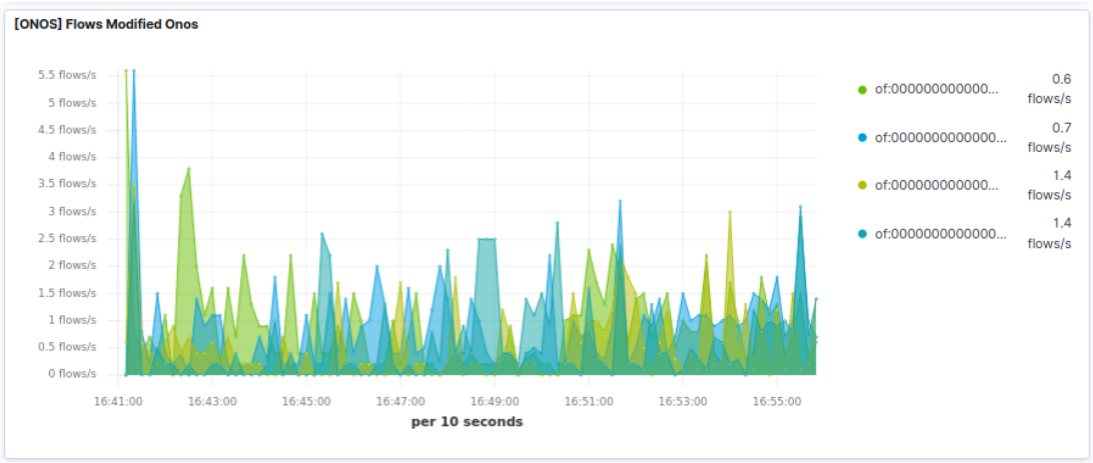


Figura 5.23: Kibana: Flujos modificados.

Gráfico *Flujos eliminados* (Figura 5.24): gráfico temporal de líneas en el que se muestran los flujos eliminados por segundo. En este caso no se ha modificado ningún flujo durante la simulación.

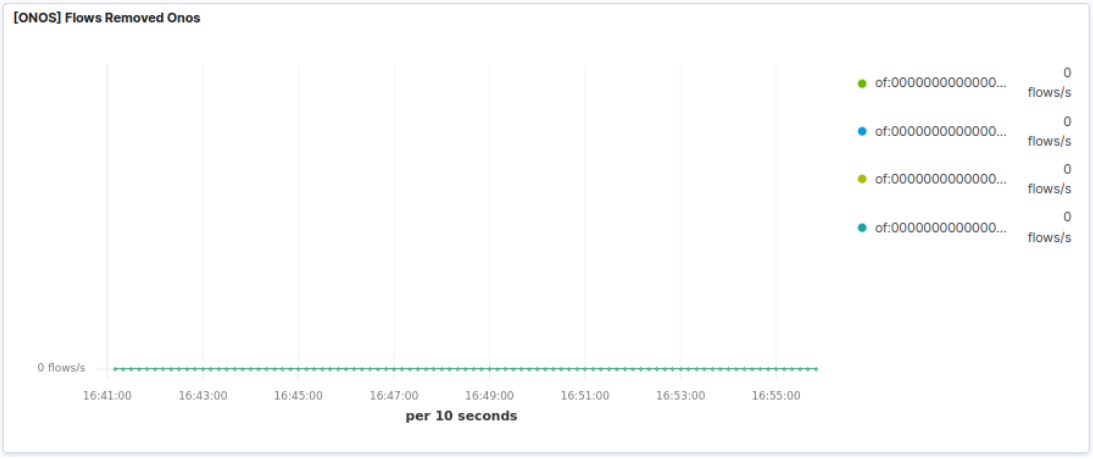


Figura 5.24: Kibana: Flujos eliminados.

Gráfico *Stats respondidos* (Figura 5.25): gráfico temporal de líneas en el que se muestran los *stats* respondidos por segundo. Al tratarse de un gráfico interactivo, desplazando el cursor sobre las diferentes líneas se mostrará el número de *stats* respondidos por segundo en ese momento para el *switch* asociado a dicha línea.

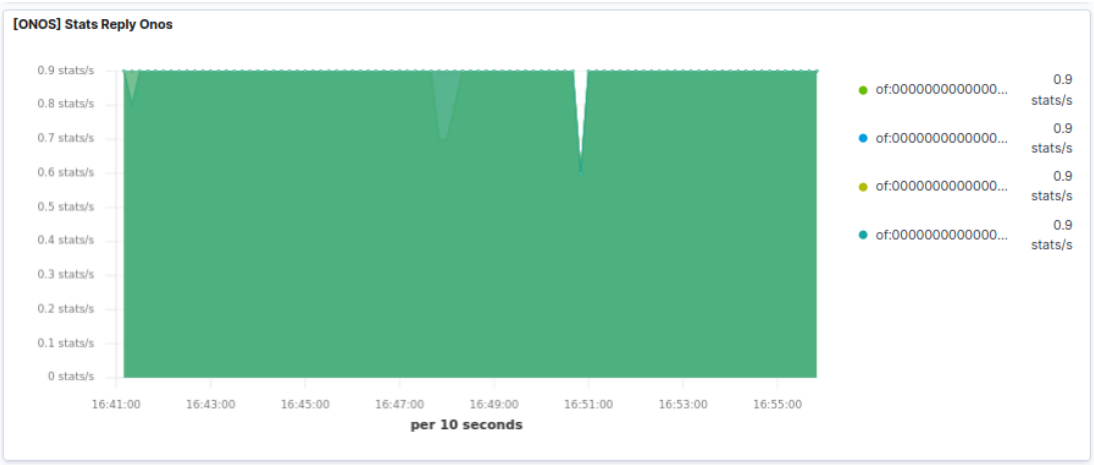


Figura 5.25: Kibana: *Stats* respondidos.

Gráfico *Stats solicitados* (Figura 5.26): gráfico temporal de líneas en el que se muestran los *stats* solicitados por segundo. Al tratarse de un gráfico interactivo, desplazando el cursor sobre las diferentes líneas se mostrará el número de *stats* solicitados por segundo en ese momento para el *switch* asociado a dicha línea.

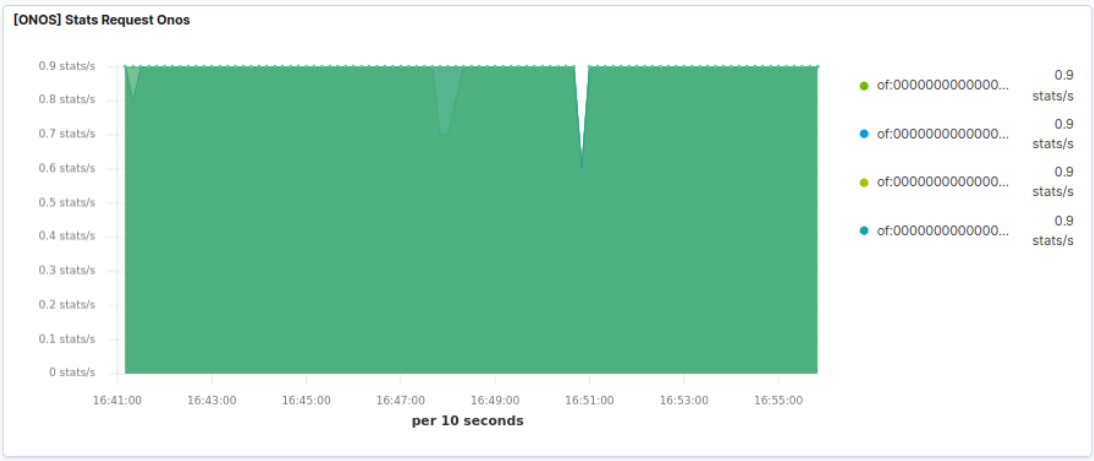
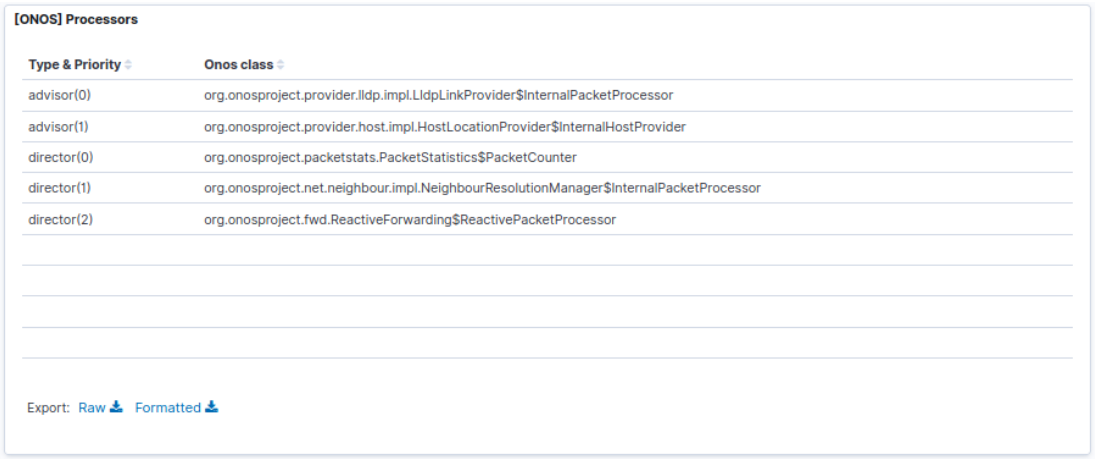


Figura 5.26: Kibana: *Stats* solicitados.

Tabla *Processors* (Figura 5.27): tabla en la que se muestran los *processors* de ONOS, su tipo, prioridad y nombre de la clase ONOS.



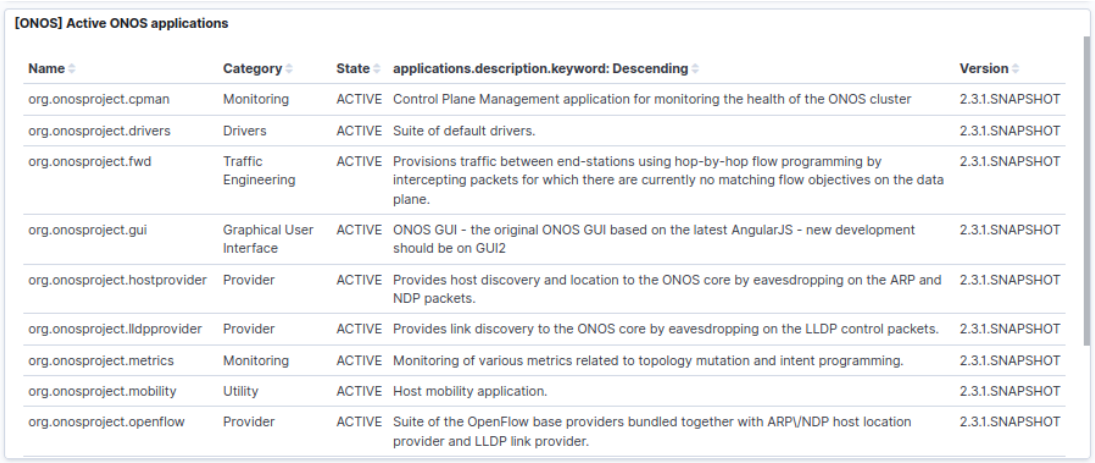
[ONOS] Processors

Type & Priority	Onos class
advisor(0)	org.onosproject.provider.lldp.impl.LldpLinkProvider$InternalPacketProcessor
advisor(1)	org.onosproject.provider.host.impl.HostLocationProvider$InternalHostProvider
director(0)	org.onosproject.packetstats.PacketStatistics$PacketCounter
director(1)	org.onosproject.net.neighbour.impl.NeighbourResolutionManager$InternalPacketProcessor
director(2)	org.onosproject.fwd.ReactiveForwarding$ReactivePacketProcessor

Export: Raw Formatted

Figura 5.27: Kibana: *Processors*.

Tabla *Aplicaciones activas* (Figura 5.28): tabla en la que se muestran las aplicaciones activas durante la ejecución de ONOS en esta simulación. Sobre ella se muestra su nombre, categoría, estado, descripción y versión.

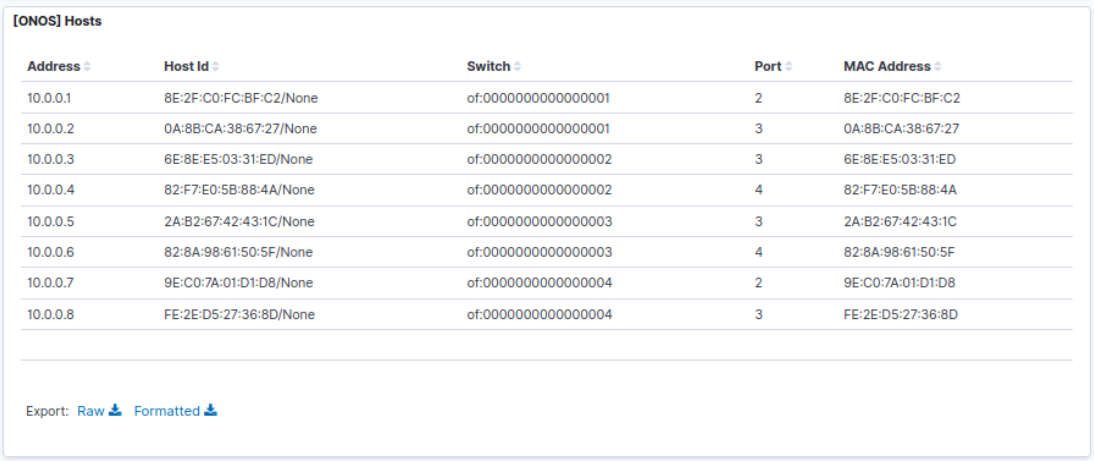


[ONOS] Active ONOS applications

Name	Category	State	applications.description.keyword: Descending	Version
org.onosproject.cpman	Monitoring	ACTIVE	Control Plane Management application for monitoring the health of the ONOS cluster	2.3.1.SNAPSHOT
org.onosproject.drivers	Drivers	ACTIVE	Suite of default drivers.	2.3.1.SNAPSHOT
org.onosproject.fwd	Traffic Engineering	ACTIVE	Provisions traffic between end-stations using hop-by-hop flow programming by intercepting packets for which there are currently no matching flow objectives on the data plane.	2.3.1.SNAPSHOT
org.onosproject.gui	Graphical User Interface	ACTIVE	ONOS GUI - the original ONOS GUI based on the latest AngularJS - new development should be on GUI2	2.3.1.SNAPSHOT
org.onosproject.hostprovider	Provider	ACTIVE	Provides host discovery and location to the ONOS core by eavesdropping on the ARP and NDP packets.	2.3.1.SNAPSHOT
org.onosproject.lldpprovider	Provider	ACTIVE	Provides link discovery to the ONOS core by eavesdropping on the LLDP control packets.	2.3.1.SNAPSHOT
org.onosproject.metrics	Monitoring	ACTIVE	Monitoring of various metrics related to topology mutation and intent programming.	2.3.1.SNAPSHOT
org.onosproject.mobility	Utility	ACTIVE	Host mobility application.	2.3.1.SNAPSHOT
org.onosproject.openflow	Provider	ACTIVE	Suite of the OpenFlow base providers bundled together with ARPI/NDP host location provider and LLDP link provider.	2.3.1.SNAPSHOT

Figura 5.28: Kibana: Aplicaciones activas.

Tabla *Hosts* (Figura 5.29): tabla en la que se muestran los *hosts* de la red. Sobre ellos se muestra: dirección IP, identificador, *switch* al que está directamente conectado, número de puerto del *switch* al que está conectado y dirección MAC.



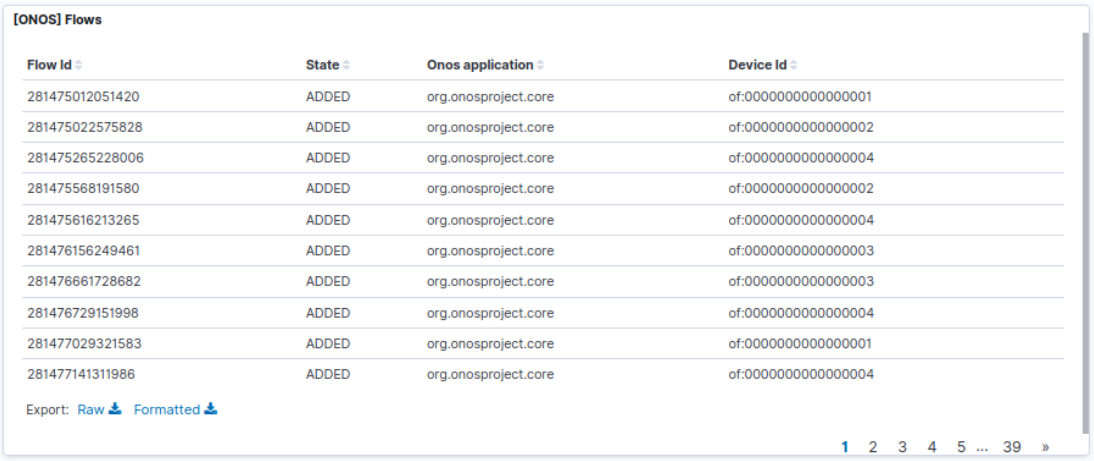
[ONOS] Hosts

Address	Host Id	Switch	Port	MAC Address
10.0.0.1	8E:2F:C0:FC:BF:C2/None	of:0000000000000001	2	8E:2F:C0:FC:BF:C2
10.0.0.2	0A:8B:CA:38:67:27/None	of:0000000000000001	3	0A:8B:CA:38:67:27
10.0.0.3	6E:8E:E5:03:31:ED/None	of:0000000000000002	3	6E:8E:E5:03:31:ED
10.0.0.4	82:F7:E0:5B:88:4A/None	of:0000000000000002	4	82:F7:E0:5B:88:4A
10.0.0.5	2A:B2:67:42:43:1C/None	of:0000000000000003	3	2A:B2:67:42:43:1C
10.0.0.6	82:8A:98:61:50:5F/None	of:0000000000000003	4	82:8A:98:61:50:5F
10.0.0.7	9E:C0:7A:01:D1:D8/None	of:0000000000000004	2	9E:C0:7A:01:D1:D8
10.0.0.8	FE:2E:D5:27:36:8D/None	of:0000000000000004	3	FE:2E:D5:27:36:8D

Export: Raw Formatted

Figura 5.29: Kibana: Hosts.

Tabla *Flujos* (Figura 5.30): tabla en la que se muestran flujos de la red. Para cada uno de los flujos se muestra su identificador, su estado, el nombre de la aplicación ONOS asociada al flujo y el nombre del *switch* asociado al flujo.



[ONOS] Flows

Flow Id	State	Onos application	Device Id
281475012051420	ADDED	org.onosproject.core	of:0000000000000001
281475022575828	ADDED	org.onosproject.core	of:0000000000000002
281475265228006	ADDED	org.onosproject.core	of:0000000000000004
281475568191580	ADDED	org.onosproject.core	of:0000000000000002
281475616213265	ADDED	org.onosproject.core	of:0000000000000004
281476156249461	ADDED	org.onosproject.core	of:0000000000000003
281476661728682	ADDED	org.onosproject.core	of:0000000000000003
281476729151998	ADDED	org.onosproject.core	of:0000000000000004
281477029321583	ADDED	org.onosproject.core	of:0000000000000001
281477141311986	ADDED	org.onosproject.core	of:0000000000000004

Export: Raw Formatted

1 2 3 4 5 ... 39 »

Figura 5.30: Kibana: Flujos.

Tras haber hecho este *dashboard* con la información general de la red, se ha creado un *dashboard* con información detallada para cada uno de los *switches*. Se puede acceder a este nuevo *dashboard* haciendo *click* en la barra del gráfico que se muestra en la Figura 5.17 asociado al *switch* del que se desee ver la información detallada. Este nuevo *dashboard* se compone de los siguientes gráficos:

Gráfico *Bytes de entrada y salida en el switch* (Figura 5.31): gráfico de tipo *gauge* en el que se muestran los bytes de entrada y salida del *switch*, tanto los bytes por segundo como los bytes totales.

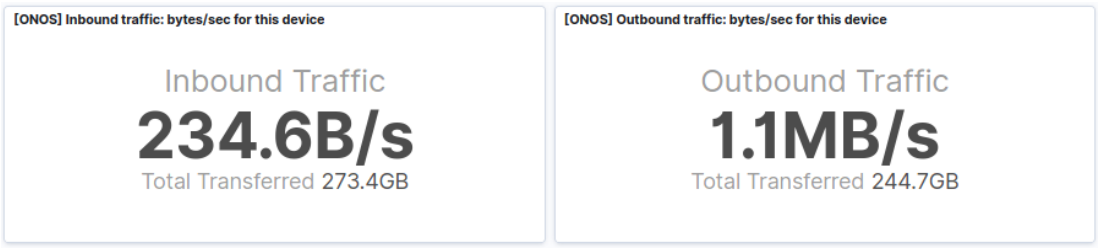


Figura 5.31: Kibana: Bytes de entrada y salida en el *switch*.

Gráfico *Paquetes de entrada y salida en el switch* (Figura 5.32): gráfico de tipo *gauge* en el que se muestran los paquetes de entrada y salida del *switch*, tanto los paquetes por segundo como los paquetes totales.

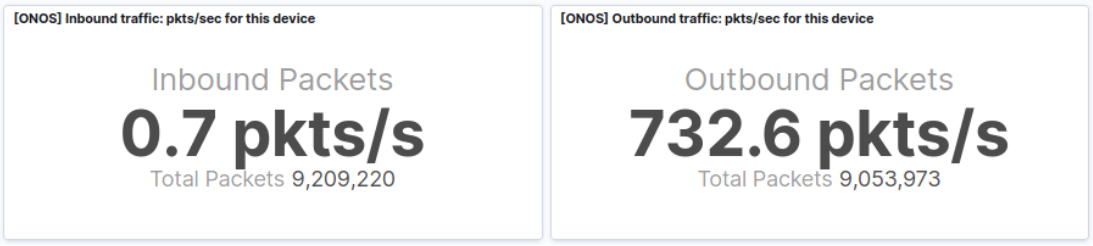


Figura 5.32: Kibana: Paquetes de entrada y salida en el *switch*.

Gráfico *Paquetes de entrada por puerto* (Figura 5.33): gráfico temporal de líneas en el que se muestran los paquetes de entrada para cada uno de los puertos del *switch*. Al tratarse de un gráfico interactivo, desplazando el cursor por las diferentes líneas del gráfico se mostrará el número de paquetes de entrada por segundo en ese momento para el puerto asociado a dicha línea.

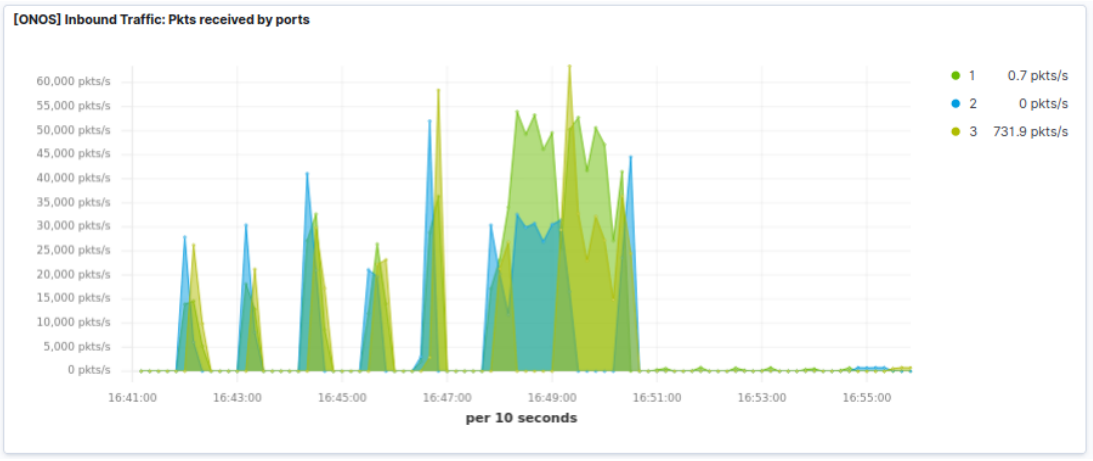


Figura 5.33: Kibana: Paquetes de entrada por puerto.

Gráfico *Paquetes de salida por puerto* (Figura 5.34): gráfico temporal de líneas en el que se muestran los paquetes de salida para cada uno de los puertos del *switch*. Al tratarse de un gráfico interactivo, desplazando el cursor por las diferentes líneas del gráfico se mostrará el número de paquetes de salida por segundo en ese momento para el puerto asociado a dicha línea.

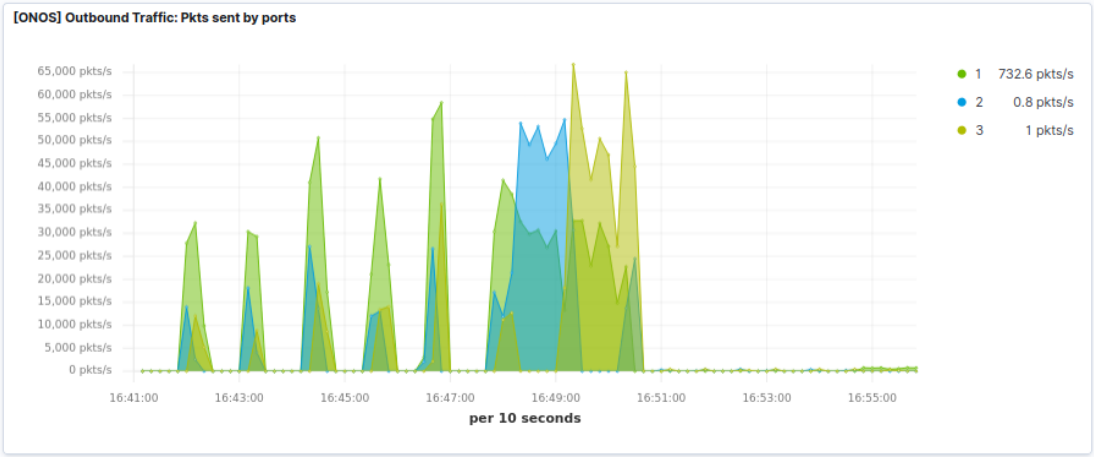


Figura 5.34: Kibana: Paquetes de salida por puerto.

Gráfico *Bytes de entrada por puerto* (Figura 5.35): gráfico temporal de líneas en el que se muestran los bytes de entrada para cada uno de los puertos del *switch*. Al tratarse de un gráfico interactivo, desplazando el cursor por las diferentes líneas del gráfico se mostrarán los bytes de entrada por segundo en ese momento para el puerto asociado a dicha línea.

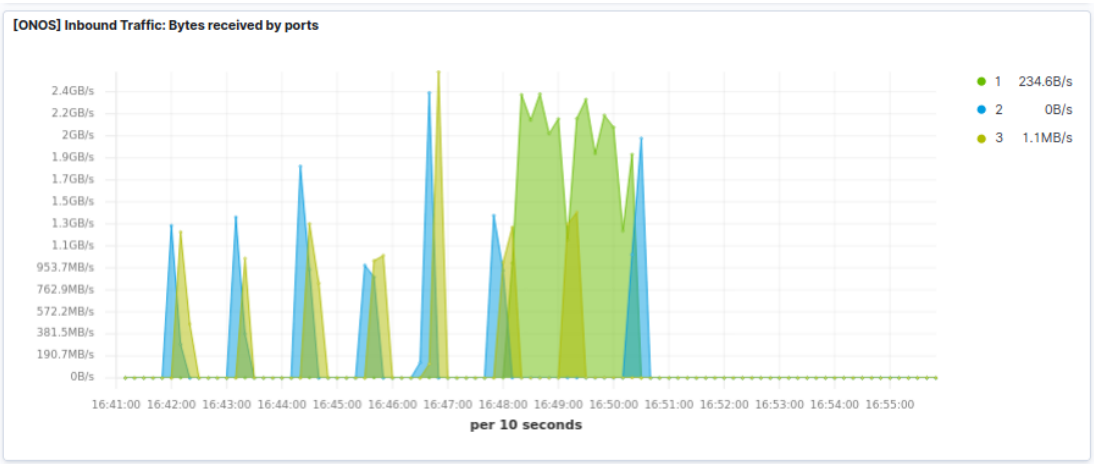


Figura 5.35: Kibana: Bytes de entrada por puerto.

Gráfico *Bytes de salida por puerto* (Figura 5.36): gráfico temporal de líneas en el que se muestran los bytes de salida para cada uno de los puertos del *switch*. Al tratarse de un gráfico

interactivo, desplazando el cursor por las diferentes líneas del gráfico se mostrarán los bytes de salida por segundo en ese momento para el puerto asociado a dicha línea.

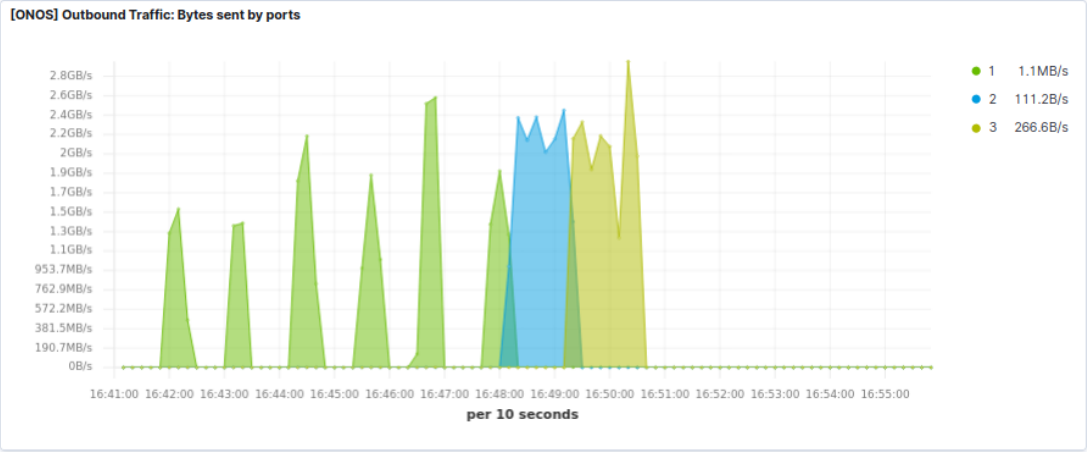


Figura 5.36: Kibana: Bytes de salida por puerto.

Tabla *Puertos del switch* (Figura 5.37): tabla en la que se muestran los puertos del *switch*. Para cada uno de los puertos se muestra el número de puerto, el número de paquetes recibidos, el número de paquetes enviados, el número de paquetes perdidos y el número de paquetes erróneos.

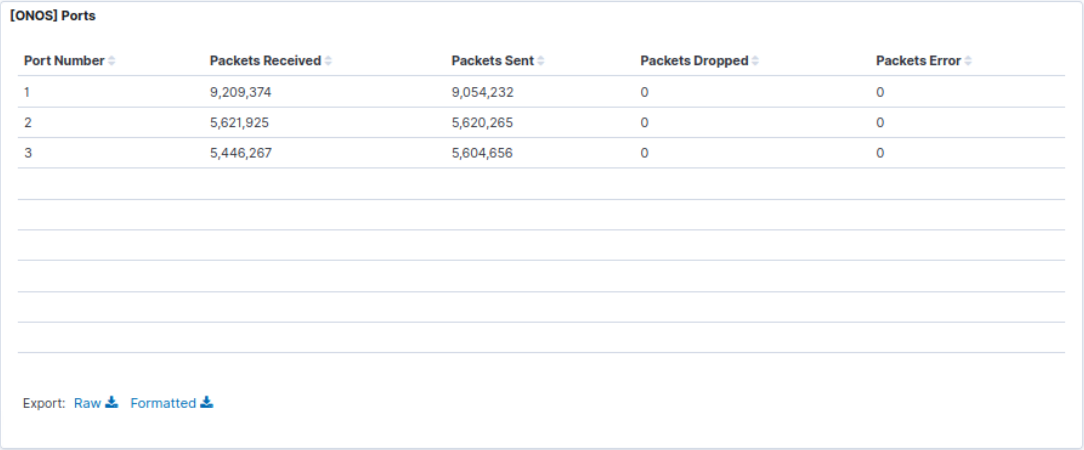


Figura 5.37: Kibana: Puertos del *switch*.

Por último, se ha creado un *dashboard* para mostrar la información de los *logs* de ONOS de forma ordenada, en este caso según su *loglevel* (INFO, WARN y ERROR). Esto puede verse en la Figura 5.38.

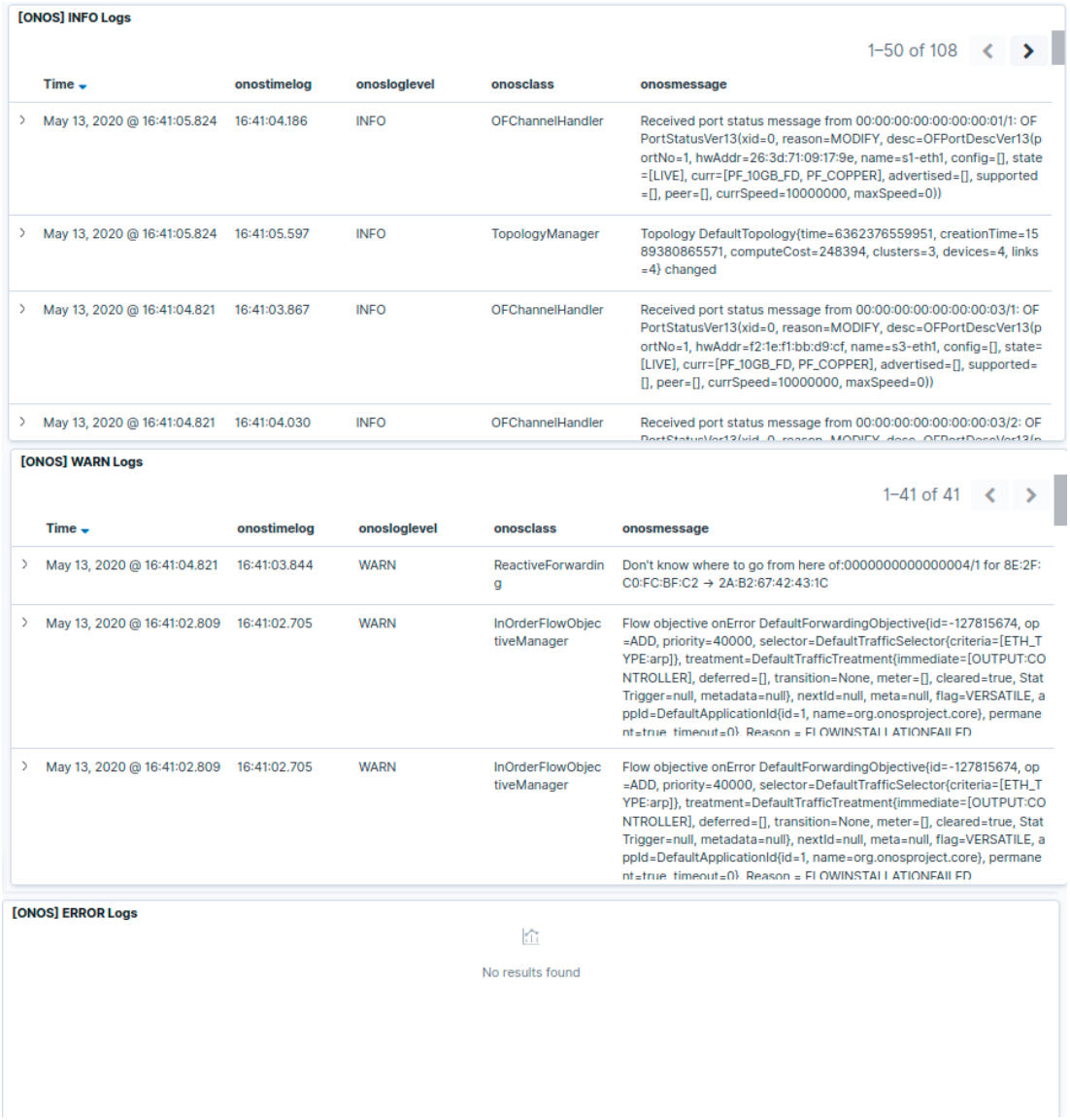


Figura 5.38: Kibana: *Dashboard* de logs.

5.6 Elastiflow

Elastiflow [14] es una herramienta *open-source* que proporciona información sobre los flujos que atraviesan la red y aporta una serie de *dashboards* fácilmente importables a Kibana. Soporta *NetFlow* v5/v9, *sFlow* e *IPFIX*. Mediante el uso de esta herramienta hemos recogido más información sobre los flujos que recorren la red emulada. El uso de esta herramienta incentivó también la utilización de Kibana como herramienta de visualización, ya que los *dashboards* que proporciona solo pueden importarse a Kibana. En concreto, permite importar

los siguientes *dashboards*:

- *Overview*: muestra la información general a modo de resumen.
- *Top-N*: muestra una serie de gráficas de tipo *TopN* acerca de los servicios, clientes, aplicaciones, etc.
- *Threats*: Elastiflow incluye un diccionario de direcciones IP públicas conocidas por su baja reputación. Este diccionario se construye a partir de muchas fuentes de datos OSINT (Open Source INTelligence), normalizadas a una taxonomía común. En este *dashboard* se utiliza esta información acerca de la reputación IP para destacar tres tipos de amenazas (*threats*):
  - Amenazas públicas: clientes públicos con una reputación IP baja.
  - Servidores de riesgo: servidores privados que son accedidos por clientes con una reputación IP baja.
  - Clientes de riesgo: clientes privados que acceden a servidores con una reputación IP baja.
- *Flows*: son *dashboards* separados mostrando las perspectivas de Cliente/Servidor, Fuente/Destino y sistemas autónomos.
- *Geo IP*: son *dashboards* separados mostrando las perspectivas de Cliente/Servidor.
- *AS Traffic*: muestra el tráfico hacia sistemas autónomos.
- *Flow Exporters*
- *Traffic Details*
- *Flow Records*
- *Ziften ZFlow*: Elastiflow ha añadido soporte a IPFIX en su versión 3.4.0.

A continuación se muestran ejemplos de los *dashboards* expuestos anteriormente e importados en la instalación de la herramienta. Estos ejemplos han sido obtenidos desde Kibana durante la ejecución de la topología desarrollada en Mininet, explicada en el Capítulo 3.

En la Figura 5.39 se muestra el *dashboard* principal de Elastiflow. En el que puede verse información relativa a los servidores, clientes, servicios, versiones IP y protocolos entre otros.

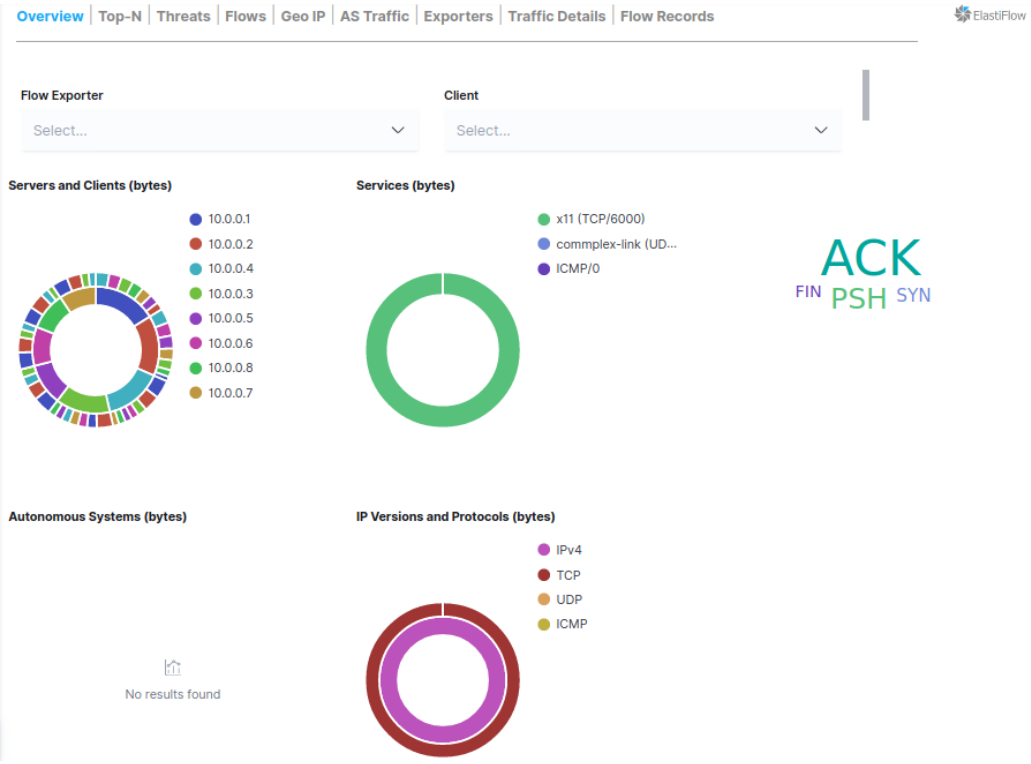
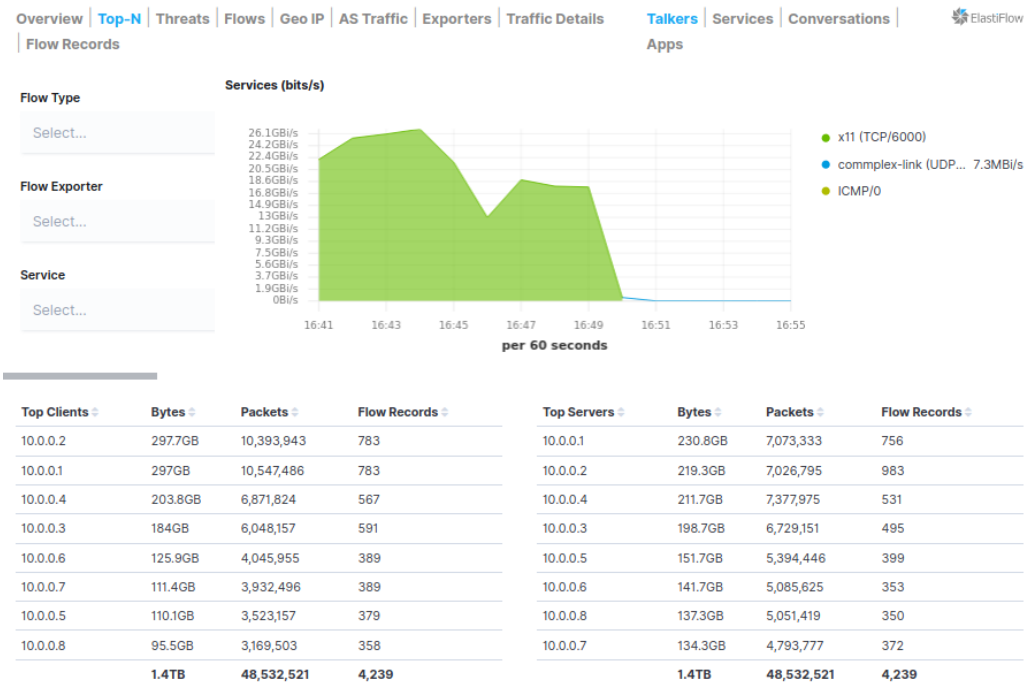


Figura 5.39: Elastiflow: *Overview*.



Top Clients	Bytes	Packets	Flow Records
10.0.0.2	297.7GB	10,393,943	783
10.0.0.1	297GB	10,547,486	783
10.0.0.4	203.8GB	6,871,824	567
10.0.0.3	184GB	6,048,157	591
10.0.0.6	125.9GB	4,045,955	389
10.0.0.7	111.4GB	3,932,496	389
10.0.0.5	110.1GB	3,523,157	379
10.0.0.8	95.5GB	3,169,503	358
	1.4TB	**48,532,521**	**4,239**

Top Servers	Bytes	Packets	Flow Records
10.0.0.1	230.8GB	7,073,333	756
10.0.0.2	219.3GB	7,026,795	983
10.0.0.4	211.7GB	7,377,975	531
10.0.0.3	198.7GB	6,729,151	495
10.0.0.5	151.7GB	5,394,446	399
10.0.0.6	141.7GB	5,085,625	353
10.0.0.8	137.3GB	5,051,419	350
10.0.0.7	134.3GB	4,793,777	372
	1.4TB	**48,532,521**	**4,239**

Figura 5.40: Elastiflow: *TopN*.

En la Figura 5.40 se muestra el *dashboard TopN* de Elastiflow. En el que puede verse información acerca de los servicios, clientes, aplicaciones, etc.

En la Figura 5.41 se muestra el *dashboard Flows* de Elastiflow. En el puede verse información relativa a los flujos de los clientes y servidores de la red, su tamaño, origen y destino.

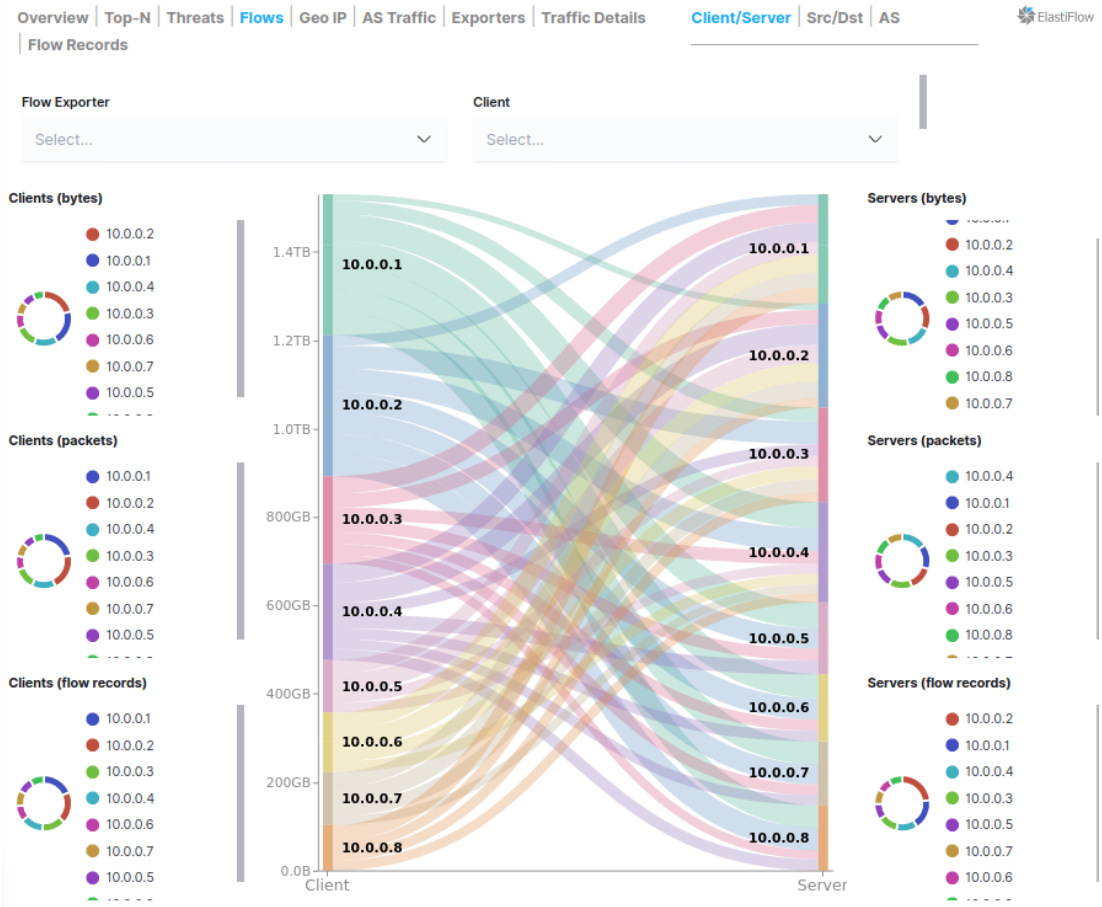


Figura 5.41: Elastiflow: *Flows*.

En la Figura 5.42 se muestra el *dashboard Exporters* de Elastiflow.

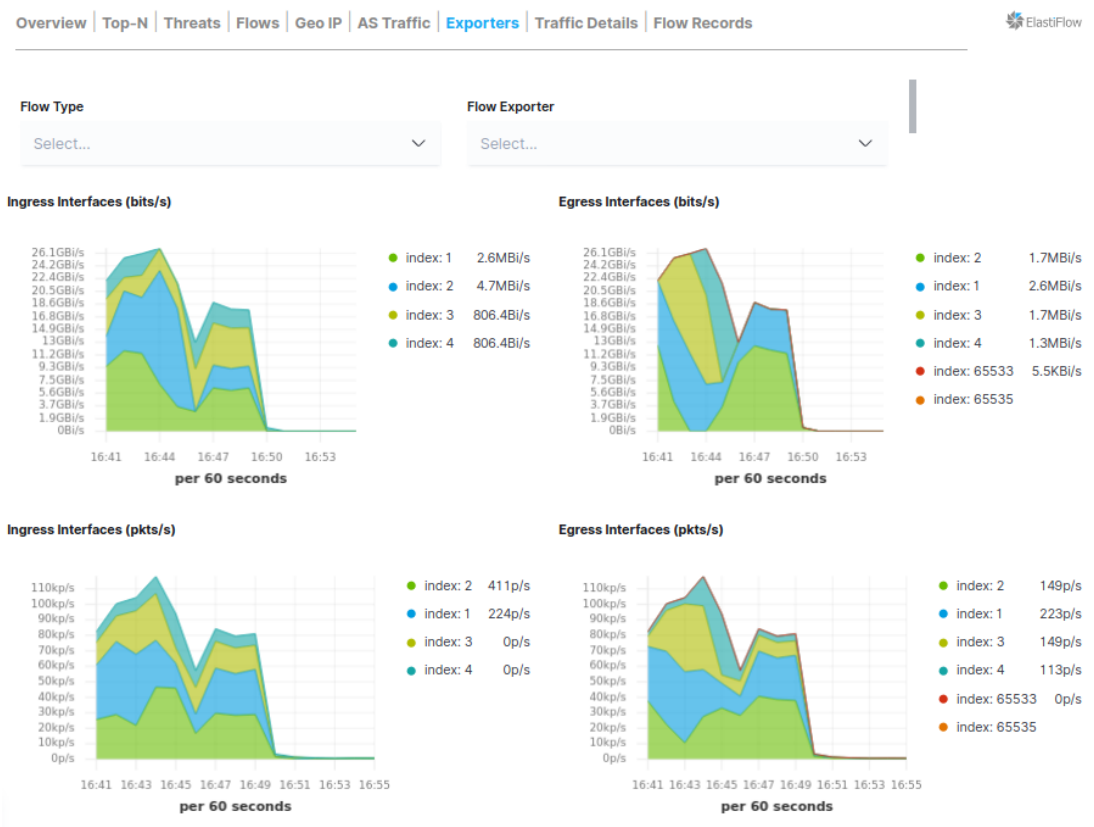


Figura 5.42: Elastiflow: *Exporters*.

En las Figuras 5.43, 5.44 y 5.45 se muestra el *dashboard Traffic Details* de Elastiflow. En el se muestra información relativa a los detalles del tráfico de red.

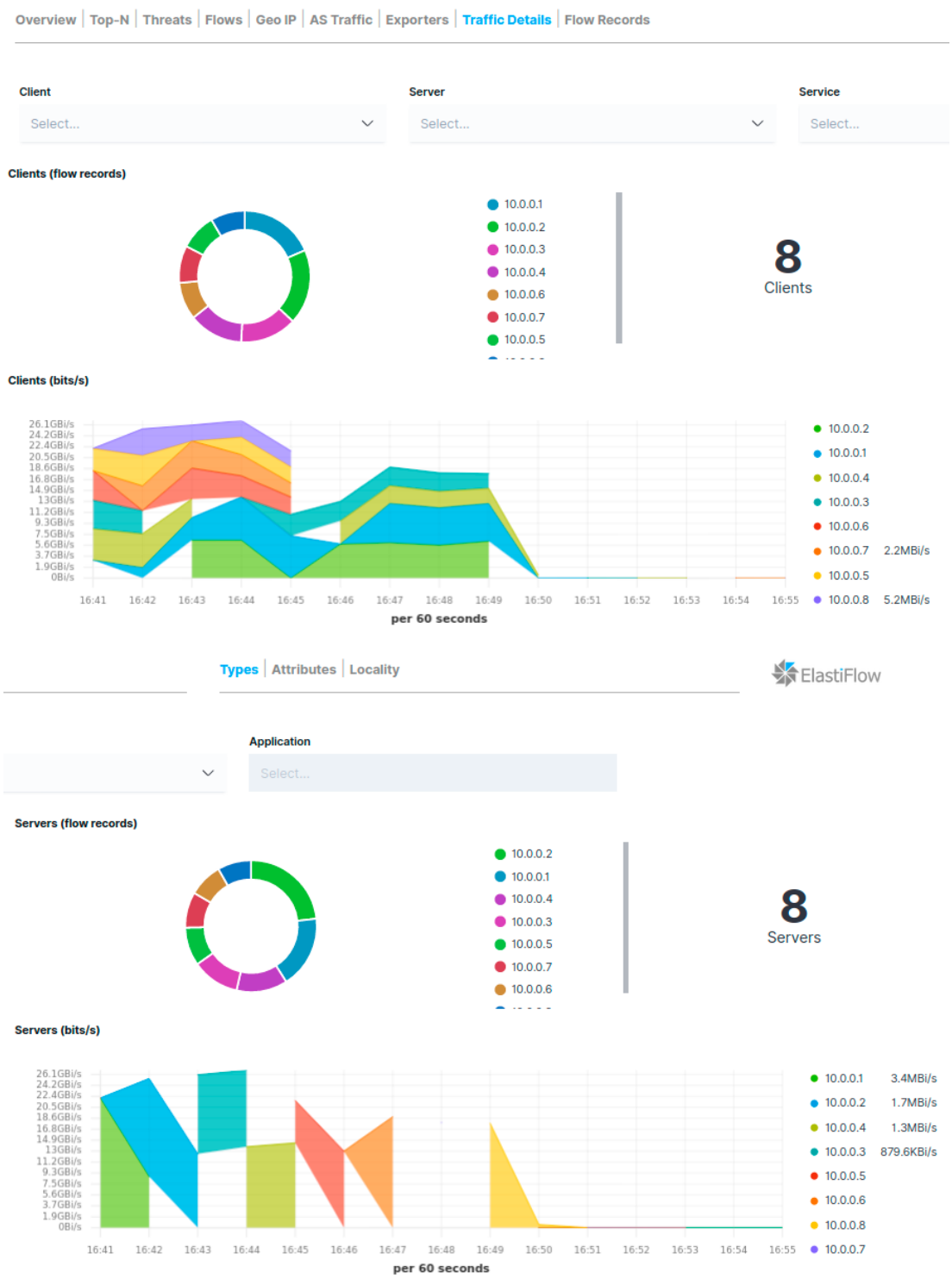


Figura 5.43: Elastiflow: *Traffic Details.*

Figura 5.44: Elastiflow: *Traffic Details.*

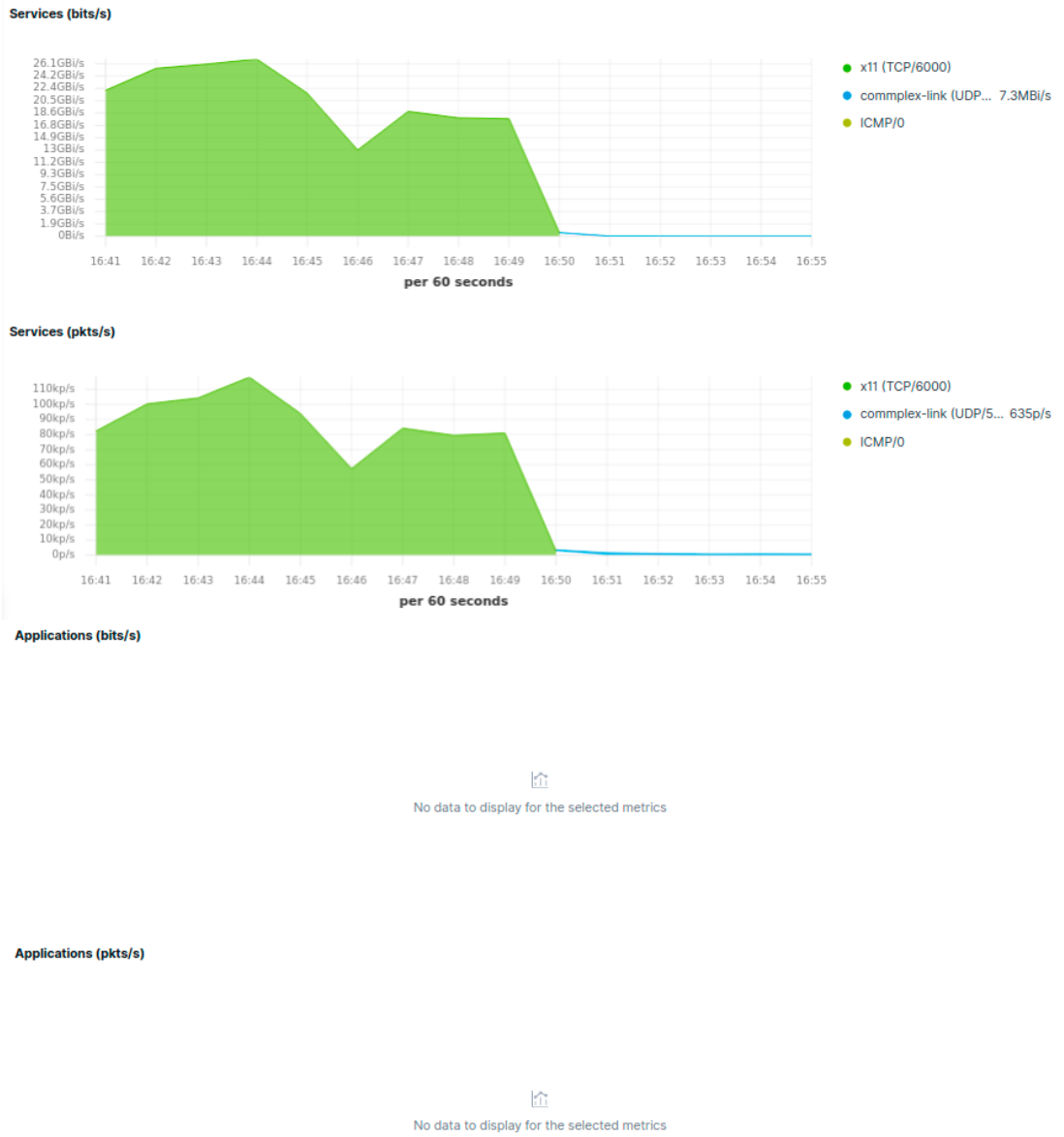


Figura 5.45: Elastiflow: *Traffic Details.*

En la Figura 5.46 se muestra el *dashboard Flow Records* de Elastiflow.

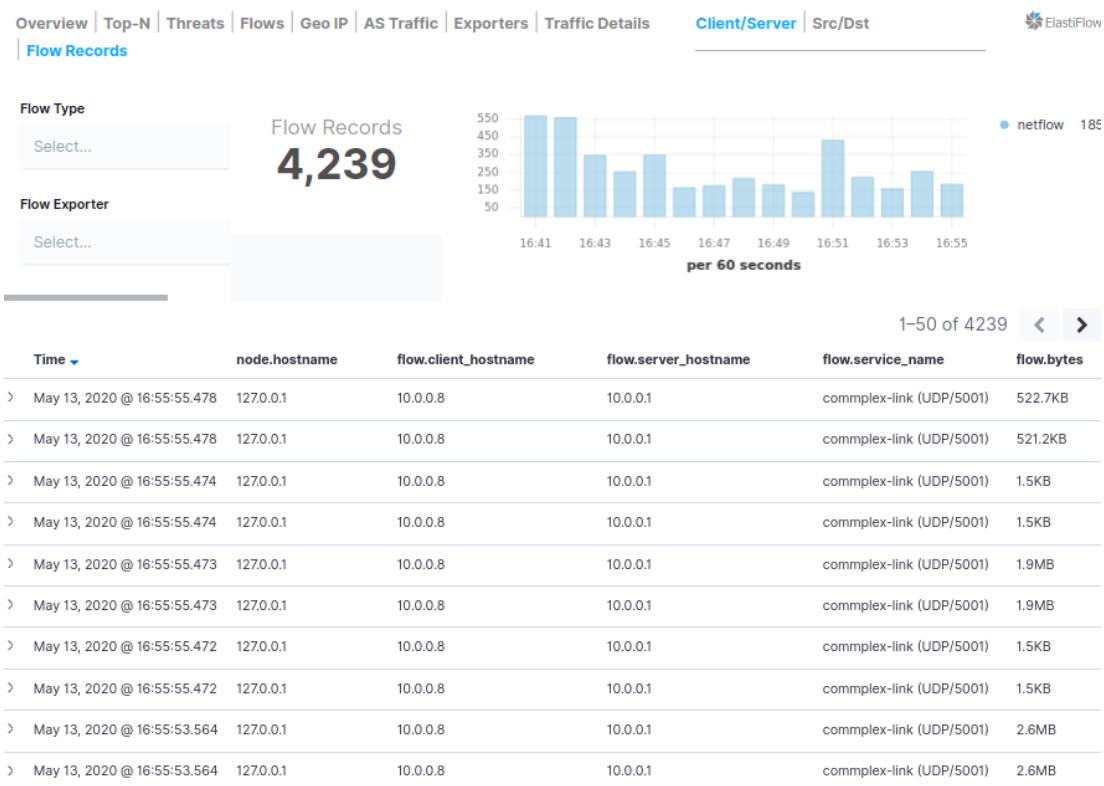


Figura 5.46: Elastiflow: *Flow Records*.

5.7 Comparativa

A continuación, tras haber creado una serie de *dashboards* utilizando las herramientas Grafana y Kibana, se expone una comparación indicando tanto los aspectos positivos como los negativos de las mismas.

5.7.1 Grafana

- Ventajas:
  - Ofrece una interfaz de usuario bastante intuitiva, facilitando en gran medida la creación de *dashboards* y gráficas, así como la de los *endpoints*.
  - Consta de un amplio catálogo de *plugins*, los cuales pueden instalarse de una forma sencilla y aumentan las funcionalidades de la herramienta, por ejemplo crear nuevos tipos de gráficas.

- Inconvenientes:
  - Hasta el momento Grafana no parece ofrecer una forma sencilla mediante la cual crear *dashboards* acerca de los ficheros de *log*. Podría hacerse un filtrado de dichos ficheros utilizando las herramientas Loki y Promtail. Sin embargo, son herramientas pensadas para el sistema operativo *OpenSuse*, por lo cual en otros sistemas operativos han de hacerse una serie de modificaciones para hacerlas funcionar correctamente. En concreto, en este proyecto no se han conseguido realizar las modificaciones pertinentes para hacerlas funcionar de la forma deseada en el sistema operativo *Ubuntu 18.04 LTS*.
  - No tiene una integración directa con la herramienta Elastiflow.

5.7.2 Kibana

- Ventajas:
  - Es muy potente, permitiendo realizar varias agregaciones, operaciones y filtrados sobre los datos.
  - Es una herramienta perteneciente a la pila ELK. Gracias a ello es posible tener todos los *dashboards*, tanto los relativos a los índices de Elasticsearch como datos filtrados a través de Logstash, en una única herramienta de visualización. En el caso particular de este proyecto, se han creado dos *dashboards* acerca de la información de la red y de los dispositivos de la misma, un *dashboard* mediante el que visualizar fácilmente los *logs* de ONOS y un último *dashboard* con información relativa a los flujos que atraviesan la red mediante la utilización de la herramienta Elastiflow.
  - Tiene una integración directa con la herramienta Elastiflow, creándose una serie de *dashboards* en Kibana automáticamente al realizar la instalación de la misma.
  - Es una herramienta que ha sido creada para tener a Elasticsearch como fuente de datos, por lo que permite realizar la recuperación de los datos rápidamente gracias a los *index patterns* de Kibana.
- Inconvenientes:
  - Es una herramienta compleja y poco intuitiva, dificultando la creación de *dashboards* y gráficas. Siendo necesario un estudio previo de la herramienta para poder utilizarla correctamente.

Capítulo 6

Conclusiones

Durante la realización de este proyecto hemos estudiado las posibilidades de monitorización de las redes definidas por software utilizando el sistema operativo de red ONOS.

Las SDNs permiten el control de la red de manera centralizada, lo que facilita en gran medida las tareas de administración, gestión y monitorización de la red. Además, gracias a su arquitectura dinámica, gestionable, adaptable y de costo eficiente, son ideales para la alta demanda de ancho de banda y para la naturaleza dinámica de las aplicaciones actuales.

Con el fin de recrear el comportamiento de una red SDN, hemos utilizado Mininet como herramienta de emulación de red. Hemos desarrollado una aplicación que, haciendo uso de los servicios y aplicaciones existentes en ONOS, nos ha permitido extraer información de interés acerca de la red emulada. Esta aplicación nos ha permitido exportar la información y métricas obtenidas a la herramienta de monitorización Prometheus, almacenarlas de manera persistente en Elasticsearch y visualizarlas utilizando Kibana. También hemos utilizado Logstash para realizar el filtrado de *logs* de ONOS y para el sondeo y procesado de su API REST, añadiendo más información a las visualizaciones creadas en Kibana. Adicionalmente, hemos explorado las posibilidades de visualización de Grafana como alternativa a Kibana.

De esta forma, hemos cumplido los objetivos propuestos en un principio. Hemos conseguido emular de manera eficiente una red SDN y realizar una monitorización completa de la misma gracias a la aplicación desarrollada y a los *dashboards* creados.

6.1 Dificultades encontradas

La realización de este trabajo ha sido realmente compleja ya que se partía desde cero, sin nociones de cómo utilizar ninguna de las herramientas empleadas.

En primer lugar, hasta el momento no existe documentación acerca de cómo desarrollar aplicaciones en las versiones más recientes de ONOS. Los únicos puntos de referencia para el estudio del funcionamiento de ONOS y de sus aplicaciones han sido la *wiki* oficial de la herramienta [7], el código fuente de la herramienta y sus aplicaciones [19] y un foro de desarrolladores. Dado que la documentación resulta insuficiente en muchos casos, hubo que recurrir a inspeccionar directamente y con exhaustividad el código fuente de ONOS y sus aplicaciones. Cabe destacar que parte de la documentación existente en la *wiki* oficial está muy obsoleta, ya que hace referencia al desarrollo y compilación de aplicaciones utilizando Maven, aún tras la migración de la herramienta a Bazel. Además, algunas de las aplicaciones existentes en el código fuente de ONOS, que sirvieron de ejemplo y guía durante el desarrollo de este proyecto, no funcionaban correctamente (bajo las condiciones, configuraciones y versiones utilizadas en este trabajo). Por ello, se tuvo que realizar un estudio en profundidad de las aplicaciones para encontrar y corregir los errores que tenían y trasladar este conocimiento a la aplicación desarrollada posteriormente en este trabajo.

En segundo lugar, comenzar a utilizar las herramientas de la pila ELK sin haber trabajado previamente con nada similar es bastante complicado. A pesar de que la documentación de Elastic está actualizada, es bastante escueta en ciertos aspectos, haciendo necesario la exploración de las herramientas y la prueba y error para finalmente aprender a utilizar correctamente las mismas.

Finalmente, a pesar de que el resto de herramientas utilizadas no han impuesto dificultades tan destacables como las nombradas anteriormente, cabe destacar el tiempo de estudio y pruebas empleado en aprender a utilizarlas para los fines propuestos en este proyecto.

6.2 Lecciones aprendidas

A pesar de las dificultades expuestas, este trabajo ha supuesto el aprendizaje de múltiples herramientas y tecnologías, además del desarrollo de una aplicación que podría aportar una nueva funcionalidad a la herramienta ONOS.

De entre las diferentes lecciones aprendidas y competencias adquiridas durante el desarrollo de este proyecto, cabe destacar el aprendizaje acerca de ONOS, su funcionamiento y cómo desarrollar aplicaciones utilizando esta herramienta; la creación de gráficos y *dashboards* que permitan realizar una monitorización correcta y completa de una red; el análisis y procesado de ficheros de *logs* y métricas utilizando Logstash, trabajando con mensajes de *logs* compuestos por varias líneas, etc. En definitiva, el aprendizaje del uso de las herramientas expuestas en este trabajo, siendo la mayoría de ellas bastante punteras y reconocidas.

6.3 Trabajo futuro

- Empaquetado en contenedores (p. ej. Docker) del trabajo realizado para facilitar su uso a terceras personas.
- Solicitación de una *pull request* a ONOS con el fin de añadir la aplicación desarrollada al conjunto de herramientas existentes en el código fuente de ONOS.
- Solicitación de *pull requests* a ONOS con el fin de mejorar la documentación existente en la *wiki* con los conocimientos adquiridos durante el desarrollo de este trabajo.

Apéndices

Apéndice A

Instalación y configuración

En este capítulo se indican las versiones de las herramientas instaladas y los pasos a seguir en su instalación y configuración.

A.1 ONOS

Se ha instalado la versión 2.3 de ONOS descargando la *release* correspondiente en su página de Github [19].

Siguiendo la guía "Quick Start" para desarrolladores, disponible en la *wiki* oficial de ONOS, se han instalado las dependencias necesarias para su correcto funcionamiento: git, zip, curl, unzip, Python 2.7, Python3, bzip2.

Por último, se ha modificado el fichero ".bashrc" añadiendo lo siguiente, siendo "ONOS_ROOT" la ruta a la carpeta en la que reside el código fuente de ONOS y "ONOS_APP" las aplicaciones que se quiere que sean activadas al lanzar ONOS:

```
#ONOS
export ONOS_ROOT=~/onos
export ONOS_APPS="gui,drivers,openflow,fwd,proxyarp,mobility"
source $ONOS_ROOT/tools/dev/bash_profile
```

A.2 Bazel

Se ha instalado la versión 1.2.1, correspondiente a Noviembre de 2019. Se ha optado por emplear esta versión dado que al realizar la instalación de versiones más actuales y probar a utilizarlas para compilar ONOS se producía un error, imposibilitando el funcionamiento del mismo y por lo tanto el desarrollo de la aplicación de monitorización SDN, objetivo de este trabajo.

Se ha seguido la guía de instalación oficial de Bazel [23]. Se ha realizado la instalación utilizando el binario obtenido de la descarga de la *release* 1.2.1 de la página de Github de Bazel [24]. Además, es necesario instalar las siguientes dependencias: "g++", "unzip", "zip", indicadas en la guía. [23]

A.3 Mininet

Se ha realizado la instalación de la versión 2.2.2 siguiendo la guía de instalación existente en la página de Github de Mininet [25]. Se ha realizado una instalación en local como paquete Ubuntu, correspondiente a la segunda opción explicada en la guía.

A.4 Prometheus

Se ha instalado la versión de Prometheus 2.1.0 como paquete Ubuntu.

Una vez hecho esto se ha modificado el fichero *prometheus.yml*, existente en la ruta */etc/-prometheus*, para cambiar el *scrape_interval* a 5s y añadir un *"job"* de modo que Prometheus pueda recolectar los datos de la aplicación ONOS desarrollada. En concreto se ha añadido el siguiente código referente al nuevo *"job"*:

```
- job_name: onos
# If prometheus-node-exporter is installed, grab stats about the
 local
# machine by default.
static_configs:
  - targets: ['localhost:1234']
```

A.5 Elasticsearch

Se ha realizado la instalación de la versión de Elasticsearch 7.6.2 siguiendo la guía oficial de Elastic [26]. En concreto, se ha optado por realizar una instalación de Elasticsearch como paquete Ubuntu.

A.6 Logstash

Se ha realizado la instalación de la versión de Logstash 7.6.2 siguiendo la guía oficial de Elastic [27]. En concreto, se ha optado por realizar una instalación de Logstash como paquete Ubuntu.

A.7 Metricbeat

Se ha realizado la instalación de la versión de Metricbeat 7.6.2 siguiendo la guía oficial de Elastic [28]. En concreto, se ha optado por realizar una instalación de Metricbeat como paquete Ubuntu.

Una vez hecho esto, se ha activado el módulo de Metricbeat relativo a Prometheus.

```
metricbeat modules enable prometheus
```

A.8 Kibana

Se ha realizado la instalación de la versión de Kibana 7.6.2 siguiendo la guía oficial de Elastic [29]. En concreto, se ha optado por realizar una instalación de Kibana como paquete Ubuntu.

A.9 Elastiflow

Se ha realizado la instalación de Elastiflow en su versión 3.5 siguiendo la guía de instalación de su página de Github [14].

A.10 Grafana

Se ha realizado la instalación de Grafana en su versión 6.7.2 como paquete Ubuntu siguiendo la guía de instalación existente en la página oficial de la herramienta [30] [31].

Una vez hecho esto, se ha instalado el *plugin Pie Chart*, siguiendo las directrices explicadas en [32], para poder crear este tipo de gráficas.

Apéndice B

Script Mininet

En este capítulo se adjunta el código del script Python de Mininet utilizado para la emulación de la red.

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from functools import partial
from mininet.node import Controller, RemoteController, OVSSwitch
from mininet.cli import CLI
from time import time, sleep

class MyTopo(Topo):
    "4 switches, 2 hosts per switch."
    def build(self):
        s1 = self.addSwitch('s1')
  s2 = self.addSwitch('s2')
  s3 = self.addSwitch('s3')
  s4 = self.addSwitch('s4')
  h1 = self.addHost('h1')
  h2 = self.addHost('h2')
  h3 = self.addHost('h3')
  h4 = self.addHost('h4')
  h5 = self.addHost('h5')
  h6 = self.addHost('h6')
  h7 = self.addHost('h7')
  h8 = self.addHost('h8')

  self.addLink(s1,s2)
  self.addLink(s2,s3)
  self.addLink(s3,s4)
```

```
31    self.addLink(h1,s1)
32    self.addLink(h2,s1)
33    self.addLink(h3,s2)
34    self.addLink(h4,s2)
35    self.addLink(h5,s3)
36    self.addLink(h6,s3)
37    self.addLink(h7,s4)
38    self.addLink(h8,s4)
39
40  def simpleTest():
41      "Create and test the network"
42      topo = MyTopo()
43      net = Mininet( topo=topo, controller=partial( RemoteController,
        ip='192.168.0.18' ),  switch=partial( OVSSwitch, protocols='
        OpenFlow13' ) )
44      net.start()
45      s1, s2, s3, s4 = net.getNodeByName('s1', 's2', 's3', 's4')
46      s1.vsctl('set Bridge s1 netflow=@id -- --id=@id create NETFLOW
        targets=\"127.0.0.1\:2055\"')
47      s2.vsctl('set Bridge s2 netflow=@id -- --id=@id create NETFLOW
        targets=\"127.0.0.1\:2055\"')
48      s3.vsctl('set Bridge s1 netflow=@id -- --id=@id create NETFLOW
        targets=\"127.0.0.1\:2055\"')
49      s4.vsctl('set Bridge s2 netflow=@id -- --id=@id create NETFLOW
        targets=\"127.0.0.1\:2055\"')
50
51      print "Dumping host connections"
52      dumpNodeConnections(net.hosts)
53
54      print "Testing network connectivity"
55      net.pingAll()
56
57      print "Iperf: Testing bandwidth between hosts"
58
59      print "Iperf: TCP Traffic"
60      print "TCP: From every host to h1"
61      net[ 'h1' ].cmd('iperf -s -p 6000 &')
62      net[ 'h2' ].cmd('iperf -c 10.0.0.1 -p 6000')
63      net[ 'h3' ].cmd('iperf -c 10.0.0.1 -p 6000')
64      net[ 'h4' ].cmd('iperf -c 10.0.0.1 -p 6000')
65      net[ 'h5' ].cmd('iperf -c 10.0.0.1 -p 6000')
66      net[ 'h6' ].cmd('iperf -c 10.0.0.1 -p 6000')
67      net[ 'h7' ].cmd('iperf -c 10.0.0.1 -p 6000')
68      net[ 'h8' ].cmd('iperf -c 10.0.0.1 -p 6000')
69
70      print "TCP: From every host to h2"
```

```
71      net[ 'h2' ].cmd('iperf -s -p 6000 &')
72      net[ 'h1' ].cmd('iperf -c 10.0.0.2 -p 6000')
73      net[ 'h3' ].cmd('iperf -c 10.0.0.2 -p 6000')
74      net[ 'h4' ].cmd('iperf -c 10.0.0.2 -p 6000')
75      net[ 'h5' ].cmd('iperf -c 10.0.0.2 -p 6000')
76      net[ 'h6' ].cmd('iperf -c 10.0.0.2 -p 6000')
77      net[ 'h7' ].cmd('iperf -c 10.0.0.2 -p 6000')
78      net[ 'h8' ].cmd('iperf -c 10.0.0.2 -p 6000')
79
80      print "TCP: From every host to h3"
81      net[ 'h3' ].cmd('iperf -s -p 6000 &')
82      net[ 'h1' ].cmd('iperf -c 10.0.0.3 -p 6000')
83      net[ 'h2' ].cmd('iperf -c 10.0.0.3 -p 6000')
84      net[ 'h4' ].cmd('iperf -c 10.0.0.3 -p 6000')
85      net[ 'h5' ].cmd('iperf -c 10.0.0.3 -p 6000')
86      net[ 'h6' ].cmd('iperf -c 10.0.0.3 -p 6000')
87      net[ 'h7' ].cmd('iperf -c 10.0.0.3 -p 6000')
88      net[ 'h8' ].cmd('iperf -c 10.0.0.3 -p 6000')
89
90      print "TCP: From every host to h4"
91      net[ 'h4' ].cmd('iperf -s -p 6000 &')
92      net[ 'h1' ].cmd('iperf -c 10.0.0.4 -p 6000')
93      net[ 'h2' ].cmd('iperf -c 10.0.0.4 -p 6000')
94      net[ 'h3' ].cmd('iperf -c 10.0.0.4 -p 6000')
95      net[ 'h5' ].cmd('iperf -c 10.0.0.4 -p 6000')
96      net[ 'h6' ].cmd('iperf -c 10.0.0.4 -p 6000')
97      net[ 'h7' ].cmd('iperf -c 10.0.0.4 -p 6000')
98      net[ 'h8' ].cmd('iperf -c 10.0.0.4 -p 6000')
99
100     print "TCP: From every host to h5"
101     net[ 'h5' ].cmd('iperf -s -p 6000 &')
102     net[ 'h1' ].cmd('iperf -c 10.0.0.5 -p 6000')
103     net[ 'h2' ].cmd('iperf -c 10.0.0.5 -p 6000')
104     net[ 'h3' ].cmd('iperf -c 10.0.0.5 -p 6000')
105     net[ 'h4' ].cmd('iperf -c 10.0.0.5 -p 6000')
106     net[ 'h6' ].cmd('iperf -c 10.0.0.5 -p 6000')
107     net[ 'h7' ].cmd('iperf -c 10.0.0.5 -p 6000')
108     net[ 'h8' ].cmd('iperf -c 10.0.0.5 -p 6000')
109
110     print "TCP: From every host to h6"
111     net[ 'h6' ].cmd('iperf -s -p 6000 &')
112     net[ 'h1' ].cmd('iperf -c 10.0.0.6 -p 6000')
113     net[ 'h2' ].cmd('iperf -c 10.0.0.6 -p 6000')
114     net[ 'h3' ].cmd('iperf -c 10.0.0.6 -p 6000')
115     net[ 'h4' ].cmd('iperf -c 10.0.0.6 -p 6000')
116     net[ 'h5' ].cmd('iperf -c 10.0.0.6 -p 6000')
```

```
117     net[ 'h7' ].cmd('iperf -c 10.0.0.6 -p 6000')
118     net[ 'h8' ].cmd('iperf -c 10.0.0.6 -p 6000')
119
120     print "TCP: From every host to h7"
121     net[ 'h7' ].cmd('iperf -s -p 6000 &')
122     net[ 'h1' ].cmd('iperf -c 10.0.0.7 -p 6000')
123     net[ 'h2' ].cmd('iperf -c 10.0.0.7 -p 6000')
124     net[ 'h3' ].cmd('iperf -c 10.0.0.7 -p 6000')
125     net[ 'h4' ].cmd('iperf -c 10.0.0.7 -p 6000')
126     net[ 'h5' ].cmd('iperf -c 10.0.0.7 -p 6000')
127     net[ 'h6' ].cmd('iperf -c 10.0.0.7 -p 6000')
128     net[ 'h8' ].cmd('iperf -c 10.0.0.7 -p 6000')
129
130     print "TCP: From every host to h8"
131     net[ 'h8' ].cmd('iperf -s -p 6000 &')
132     net[ 'h1' ].cmd('iperf -c 10.0.0.8 -p 6000')
133     net[ 'h2' ].cmd('iperf -c 10.0.0.8 -p 6000')
134     net[ 'h3' ].cmd('iperf -c 10.0.0.8 -p 6000')
135     net[ 'h4' ].cmd('iperf -c 10.0.0.8 -p 6000')
136     net[ 'h5' ].cmd('iperf -c 10.0.0.8 -p 6000')
137     net[ 'h6' ].cmd('iperf -c 10.0.0.8 -p 6000')
138     net[ 'h7' ].cmd('iperf -c 10.0.0.8 -p 6000')
139
140
141     print "Iperf: UDP Traffic"
142     print "UDP: From h1 to every host"
143     h1, h2 = net.getNodeByName('h1', 'h2')
144     net.iperf( ( h1, h2 ), l4Type='UDP' )
145     h1, h3 = net.getNodeByName('h1', 'h3')
146     net.iperf( ( h1, h3 ), l4Type='UDP' )
147     h1, h4 = net.getNodeByName('h1', 'h4')
148     net.iperf( ( h1, h4 ), l4Type='UDP' )
149     h1, h5 = net.getNodeByName('h1', 'h5')
150     net.iperf( ( h1, h5 ), l4Type='UDP' )
151     h1, h6 = net.getNodeByName('h1', 'h6')
152     net.iperf( ( h1, h6 ), l4Type='UDP' )
153     h1, h7 = net.getNodeByName('h1', 'h7')
154     net.iperf( ( h1, h7 ), l4Type='UDP' )
155     h1, h8 = net.getNodeByName('h1', 'h8')
156     net.iperf( ( h1, h8 ), l4Type='UDP' )
157
158     print "UDP: From h2 to every host"
159     h2, h1 = net.getNodeByName('h2', 'h1')
160     net.iperf( ( h2, h1 ), l4Type='UDP' )
161     h2, h3 = net.getNodeByName('h2', 'h3')
162     net.iperf( ( h2, h3 ), l4Type='UDP' )
```

```
163     h2, h4 = net.getNodeByName('h2', 'h4')
164     net.iperf( ( h2, h4 ), l4Type='UDP' )
165     h2, h5 = net.getNodeByName('h2', 'h5')
166     net.iperf( ( h2, h5 ), l4Type='UDP' )
167     h2, h6 = net.getNodeByName('h2', 'h6')
168     net.iperf( ( h2, h6 ), l4Type='UDP' )
169     h2, h7 = net.getNodeByName('h2', 'h7')
170     net.iperf( ( h2, h7 ), l4Type='UDP' )
171     h2, h8 = net.getNodeByName('h2', 'h8')
172     net.iperf( ( h2, h8 ), l4Type='UDP' )
173
174     print "UDP: From h3 to every host"
175     h3, h2 = net.getNodeByName('h3', 'h2')
176     net.iperf( ( h3, h2 ), l4Type='UDP' )
177     h3, h1 = net.getNodeByName('h3', 'h1')
178     net.iperf( ( h3, h1 ), l4Type='UDP' )
179     h3, h4 = net.getNodeByName('h3', 'h4')
180     net.iperf( ( h3, h4 ), l4Type='UDP' )
181     h3, h5 = net.getNodeByName('h3', 'h5')
182     net.iperf( ( h3, h5 ), l4Type='UDP' )
183     h3, h6 = net.getNodeByName('h3', 'h6')
184     net.iperf( ( h3, h6 ), l4Type='UDP' )
185     h3, h7 = net.getNodeByName('h3', 'h7')
186     net.iperf( ( h3, h7 ), l4Type='UDP' )
187     h3, h8 = net.getNodeByName('h3', 'h8')
188     net.iperf( ( h3, h8 ), l4Type='UDP' )
189
190     print "UDP: From h4 to every host"
191     h4, h2 = net.getNodeByName('h4', 'h2')
192     net.iperf( ( h4, h2 ), l4Type='UDP' )
193     h4, h3 = net.getNodeByName('h4', 'h3')
194     net.iperf( ( h4, h3 ), l4Type='UDP' )
195     h4, h1 = net.getNodeByName('h4', 'h1')
196     net.iperf( ( h4, h1 ), l4Type='UDP' )
197     h4, h5 = net.getNodeByName('h4', 'h5')
198     net.iperf( ( h4, h5 ), l4Type='UDP' )
199     h4, h6 = net.getNodeByName('h4', 'h6')
200     net.iperf( ( h4, h6 ), l4Type='UDP' )
201     h4, h7 = net.getNodeByName('h4', 'h7')
202     net.iperf( ( h4, h7 ), l4Type='UDP' )
203     h4, h8 = net.getNodeByName('h4', 'h8')
204     net.iperf( ( h4, h8 ), l4Type='UDP' )
205
206     print "UDP: From h5 to every host"
207     h5, h2 = net.getNodeByName('h5', 'h2')
208     net.iperf( ( h1, h2 ), l4Type='UDP' )
```

```
    h5, h3 = net.getNodeByName('h5', 'h3')
    net.iperf( ( h5, h3 ), l4Type='UDP' )
    h5, h4 = net.getNodeByName('h5', 'h4')
    net.iperf( ( h5, h4 ), l4Type='UDP' )
    h5, h1 = net.getNodeByName('h5', 'h1')
    net.iperf( ( h5, h1 ), l4Type='UDP' )
    h5, h6 = net.getNodeByName('h5', 'h6')
    net.iperf( ( h5, h6 ), l4Type='UDP' )
    h5, h7 = net.getNodeByName('h5', 'h7')
    net.iperf( ( h5, h7 ), l4Type='UDP' )
    h5, h8 = net.getNodeByName('h5', 'h8')
    net.iperf( ( h5, h8 ), l4Type='UDP' )

    print "UDP: From h6 to every host"
    h6, h2 = net.getNodeByName('h6', 'h2')
    net.iperf( ( h6, h2 ), l4Type='UDP' )
    h6, h3 = net.getNodeByName('h6', 'h3')
    net.iperf( ( h6, h3 ), l4Type='UDP' )
    h6, h4 = net.getNodeByName('h6', 'h4')
    net.iperf( ( h6, h4 ), l4Type='UDP' )
    h6, h5 = net.getNodeByName('h6', 'h5')
    net.iperf( ( h6, h5 ), l4Type='UDP' )
    h6, h1 = net.getNodeByName('h6', 'h1')
    net.iperf( ( h6, h1 ), l4Type='UDP' )
    h6, h7 = net.getNodeByName('h6', 'h7')
    net.iperf( ( h6, h7 ), l4Type='UDP' )
    h6, h8 = net.getNodeByName('h6', 'h8')
    net.iperf( ( h6, h8 ), l4Type='UDP' )

    print "UDP: From h7 to every host"
    h7, h2 = net.getNodeByName('h7', 'h2')
    net.iperf( ( h7, h2 ), l4Type='UDP' )
    h7, h3 = net.getNodeByName('h7', 'h3')
    net.iperf( ( h7, h3 ), l4Type='UDP' )
    h7, h4 = net.getNodeByName('h7', 'h4')
    net.iperf( ( h7, h4 ), l4Type='UDP' )
    h7, h5 = net.getNodeByName('h7', 'h5')
    net.iperf( ( h7, h5 ), l4Type='UDP' )
    h7, h1 = net.getNodeByName('h7', 'h1')
    net.iperf( ( h7, h1 ), l4Type='UDP' )
    h7, h6 = net.getNodeByName('h7', 'h6')
    net.iperf( ( h7, h6 ), l4Type='UDP' )
    h7, h8 = net.getNodeByName('h7', 'h8')
    net.iperf( ( h7, h8 ), l4Type='UDP' )

    print "UDP: From h8 to every host"
```

```
255     h8, h2 = net.getNodeByName('h8', 'h2')
256     net.iperf( ( h8, h2 ), l4Type='UDP' )
257     h8, h3 = net.getNodeByName('h8', 'h3')
258     net.iperf( ( h8, h3 ), l4Type='UDP' )
259     h8, h4 = net.getNodeByName('h8', 'h4')
260     net.iperf( ( h8, h4 ), l4Type='UDP' )
261     h8, h5 = net.getNodeByName('h8', 'h5')
262     net.iperf( ( h8, h5 ), l4Type='UDP' )
263     h8, h1 = net.getNodeByName('h8', 'h1')
264     net.iperf( ( h8, h1 ), l4Type='UDP' )
265     h8, h7 = net.getNodeByName('h8', 'h7')
266     net.iperf( ( h8, h7 ), l4Type='UDP' )
267     h8, h6 = net.getNodeByName('h8', 'h6')
268     net.iperf( ( h8, h6 ), l4Type='UDP' )
269
270     CLI(net)
271     net.stop()
272
273 if __name__ == '__main__':
274     # Tell mininet to print useful information
275     setLogLevel('info')
276     simpleTest()
```

Listing B.1: Script Mininet.

Apéndice C

Aplicación ONOS

En este capítulo se adjunta el código de la aplicación Java desarrollada en ONOS para la exportación de métricas.

```
1  package org.onosproject.prometheuscollector;
2
3  import org.onlab.metrics.MetricsService;
4  import org.onosproject.core.ApplicationId;
5  import org.onosproject.core.CoreService;
6  import org.osgi.service.component.annotations.Activate;
7  import org.osgi.service.component.annotations.Component;
8  import org.osgi.service.component.annotations.Deactivate;
9  import org.osgi.service.component.annotations.Reference;
10 import org.osgi.service.component.annotations.ReferenceCardinality;
11 import org.slf4j.Logger;
12 import java.util.ArrayList;
13 import java.util.Arrays;
14 import java.util.List;
15 import java.util.Map;
16 import java.util.HashMap;
17
18 import io.prometheus.client.dropwizard.DropwizardExports;
19 import io.prometheus.client.dropwizard.samplebuilder.MapperConfig;
20 import io.prometheus.client.dropwizard.samplebuilder.
       CustomMappingSampleBuilder;
21 import io.prometheus.client.dropwizard.samplebuilder.SampleBuilder;
22 import io.prometheus.client.exporter.MetricsServlet;
23 import org.eclipse.jetty.server.Server;
24 import org.eclipse.jetty.servlet.ServletContextHandler;
25 import org.eclipse.jetty.servlet.ServletHolder;
26 import org.onosproject.net.device.DeviceService;
27 import org.onosproject.net.statistic.StatisticService;
28
29 import org.onosproject.net.Device;
```

```
30 import io.prometheus.client.Collector;
31 import io.prometheus.client.GaugeMetricFamily;
32 import org.onosproject.net.ConnectPoint;
33 import org.onosproject.net.Port;
34 import org.onosproject.net.flow.FlowRule;
35
36 import static org.slf4j.LoggerFactory.getLogger;
37
38
39 @Component(immediate = true)
40 public class Main {
41     private static final Logger log = getLogger(CustomCollector.
    class);
42
43     @Reference(cardinality = ReferenceCardinality.MANDATORY)
44     protected CoreService coreService;
45
46     @Reference(cardinality = ReferenceCardinality.MANDATORY)
47     protected MetricsService metricsService;
48
49     @Reference(cardinality = ReferenceCardinality.MANDATORY)
50     protected StatisticService statisticService;
51
52     @Reference(cardinality = ReferenceCardinality.MANDATORY)
53     protected DeviceService deviceService;
54
55     private ApplicationId appId;
56
57
58     @Activate
59     public void activate() throws Exception {
60         log.info("----ACTIVATING PROMETHEUS COLLECTOR APP----");
61         appId = coreService.registerApplication("org.onosproject.
    prometheuscollector");
62
63         //Configure Dropwizard labels
64         SampleBuilder sampleBuilder = configLabels();
65
66         // Registration
67         CustomCollector requests = new CustomCollector().register();
68         new DropwizardExports(metricsService.getMetricRegistry(),
    sampleBuilder).register();
69         log.info("----COLLECTORS REGISTERED----");
70
71         serverConfig();
72
```

```
73          log.info("--------------------------");
74      }
75
76      @Deactivate
77      public void deactivate() {
78          log.info("----DEACTIVATING PROMETHEUS COLLECTOR APP----");
79          log.info("Stopped");
80      }
81
82      public SampleBuilder configLabels() {
83          //Configure Dropwizard labels - cpman app related
84          MapperConfig config1 = new MapperConfig();
85          config1.setMatch("*.FLOW_MOD.count");
86          config1.setName("FLOW_MOD.count");
87          Map<String, String> labels1 = new HashMap<String, String>();
88          labels1.put("deviceId", "${0}");
89          config1.setLabels(labels1);
90
91          MapperConfig config2 = new MapperConfig();
92          config2.setMatch("*.FLOW_MOD.rate");
93          config2.setName("FLOW_MOD.rate");
94          Map<String, String> labels2 = new HashMap<String, String>();
95          labels2.put("deviceId", "${0}");
96          config2.setLabels(labels2);
97
98          MapperConfig config3 = new MapperConfig();
99          config3.setMatch("*.FLOW_REMOVED.count");
100         config3.setName("FLOW_REMOVED.count");
101         Map<String, String> labels3 = new HashMap<String, String>();
102         labels3.put("deviceId", "${0}");
103         config3.setLabels(labels3);
104
105         MapperConfig config4 = new MapperConfig();
106         config4.setMatch("*.FLOW_REMOVED.rate");
107         config4.setName("FLOW_REMOVED.rate");
108         Map<String, String> labels4 = new HashMap<String, String>();
109         labels4.put("deviceId", "${0}");
110         config4.setLabels(labels4);
111
112         MapperConfig config5 = new MapperConfig();
113         config5.setMatch("*.PACKET_IN.count");
114         config5.setName("PACKET_IN.count");
115         Map<String, String> labels5 = new HashMap<String, String>();
116         labels5.put("deviceId", "${0}");
117         config5.setLabels(labels5);
118
```

```
119         MapperConfig config6 = new MapperConfig();
120         config6.setMatch("*.PACKET_IN.rate");
121         config6.setName("PACKET_IN.rate");
122         Map<String, String> labels6 = new HashMap<String, String>();
123         labels6.put("deviceId", "${0}");
124         config6.setLabels(labels6);
125 
126         MapperConfig config7 = new MapperConfig();
127         config7.setMatch("*.PACKET_OUT.count");
128         config7.setName("PACKET_OUT.count");
129         Map<String, String> labels7 = new HashMap<String, String>();
130         labels7.put("deviceId", "${0}");
131         config7.setLabels(labels7);
132 
133         MapperConfig config8 = new MapperConfig();
134         config8.setMatch("*.PACKET_OUT.rate");
135         config8.setName("PACKET_OUT.rate");
136         Map<String, String> labels8 = new HashMap<String, String>();
137         labels8.put("deviceId", "${0}");
138         config8.setLabels(labels8);
139 
140         MapperConfig config9 = new MapperConfig();
141         config9.setMatch("*.STATS_REPLY.count");
142         config9.setName("STATS_REPLY.count");
143         Map<String, String> labels9 = new HashMap<String, String>();
144         labels9.put("deviceId", "${0}");
145         config9.setLabels(labels9);
146 
147         MapperConfig config10 = new MapperConfig();
148         config10.setMatch("*.STATS_REPLY.rate");
149         config10.setName("STATS_REPLY.rate");
150         Map<String, String> labels10 = new HashMap<String, String>()
    ;
151         labels10.put("deviceId", "${0}");
152         config10.setLabels(labels10);
153 
154         MapperConfig config11 = new MapperConfig();
155         config11.setMatch("*.STATS_REQUEST.count");
156         config11.setName("STATS_REQUEST.count");
157         Map<String, String> labels11 = new HashMap<String, String>()
    ;
158         labels11.put("deviceId", "${0}");
159         config11.setLabels(labels11);
160 
161         MapperConfig config12 = new MapperConfig();
162         config12.setMatch("*.STATS_REQUEST.rate");
```

```
163         config12.setName("STATS_REQUEST.rate");
164         Map<String, String> labels12 = new HashMap<String, String>()
    ;
165         labels12.put("deviceId", "${0}");
166         config12.setLabels(labels12);
167 
168         List<MapperConfig> list = new ArrayList();
169         list.add(config1);
170         list.add(config2);
171         list.add(config3);
172         list.add(config4);
173         list.add(config5);
174         list.add(config6);
175         list.add(config7);
176         list.add(config8);
177         list.add(config9);
178         list.add(config10);
179         list.add(config11);
180         list.add(config12);
181         SampleBuilder sampleBuilder = new CustomMappingSampleBuilder
    (list);
182         return sampleBuilder;
183     }
184 
185     public void serverConfig() throws Exception {
186         // Create the server
187         Server server = new Server(1234);
188 
189         // Add ServletContextHandler
190         ServletContextHandler servletContextHandler = new
    ServletContextHandler(
191         ServletContextHandler.SESSIONS);
192         servletContextHandler.setContextPath("/");
193         server.setHandler(servletContextHandler);
194 
195         // Add MetricsServlet
196         ServletHolder holder = new ServletHolder(new MetricsServlet
    ());
197         servletContextHandler.addServlet(holder, "/metrics");
198 
199         // Start the server
200         server.start();
201         log.info("----SERVER STARTED----");
202     }
203 
204     private class CustomCollector extends Collector {
```

```
205         @Override
206         public List<MetricFamilySamples> collect() {
207             List<MetricFamilySamples> mfs = new ArrayList<
    MetricFamilySamples>();
208             GaugeMetricFamily rate = new GaugeMetricFamily("rate", "
    CP rate as bytes per second",
209             Arrays.asList("deviceId", "portNumber"));
210             GaugeMetricFamily hitter = new GaugeMetricFamily("
    highest_hitter", "CP highest hitter",
211             Arrays.asList("deviceId", "portNumber"));
212 
213             Iterable<Device> devices = deviceService.getDevices();
214 
215             for (Device d : devices) {
216                 List<Port> ports = deviceService.getPorts(d.id());
217                 for (int i = 0; i < ports.size(); i++) {
218                     ConnectPoint cp = new ConnectPoint(d.id(), ports
    .get(i).number());
219                     String dId = d.id().toString();
220                     String pNum = ports.get(i).number().toString();
221                     //Getting rate from StatisticsService
222                     rate.addMetric(Arrays.asList(dId, pNum),
    statisticService.load(cp).rate());
223                     //Getting highest hitter from StatisticsService
224                     if (statisticService.highestHitter(cp) == null)
    {
225                         hitter.addMetric(Arrays.asList(dId, pNum),
    0);
226                     } else {
227                         FlowRule fr = statisticService.highestHitter
    (cp);
228                         hitter.addMetric(Arrays.asList(dId, pNum),
    fr.id().value());
229                     }
230                     }
231             }
232             mfs.add(hitter);
233             mfs.add(rate);
234             return mfs;
235         }
236     }
237 }
```

Listing C.1: Aplicación ONOS desarrollada.

Apéndice D

Pipelines de Logstash

D.1 Pipeline para los *logs* de ONOS

```
input {
  file {
    path => "/tmp/onos-2.3.1-SNAPSHOT/onos.log"
    type => "onoslogs"
    codec => multiline {
      pattern => "^\s"
      what => "previous"
    }
  }
}
filter {
  mutate {
    gsub => ["message", "\x1B\[([0-9]{1,2}(;[0-9]{1,2})?)?[m|K]",
    ""]
  }
  grok {
    match => { "message" => "%{TIME:onostimelog}%{SPACE}%{LOGLEVEL:
    onosloglevel}%{SPACE}[\[]%{DATA:onosclass}[\]]%{SPACE}%{
    GREEDYDATA:onosmessage}" }
  }
}
output {
  if [type] == "onoslogs" {
    elasticsearch {
      hosts => ["localhost:9200"]
      index => "onoslogs-%{+YYYY.MM.dd}"
    }
  }
}
```

Listing D.1: Pipeline para el log de ONOS.

D.2 Pipeline para el API REST de ONOS

```
input {
   http_poller {
     urls => {
       onos_apps => {
         # Supports all options supported by ruby's Manticore HTTP
    client
         method => get
         user => "onos"
         password => "rocks"
         url => "http://localhost:8181/onos/v1/applications"
         headers => {
           Accept => "application/json"
         }
       }
       onos_devices => {
         method => get
         user => "onos"
         password => "rocks"
         url => "http://localhost:8181/onos/v1/devices"
         headers => {
           Accept => "application/json"
         }
       }
       onos_flows => {
         method => get
         user => "onos"
         password => "rocks"
         url => "http://localhost:8181/onos/v1/flows"
         headers => {
           Accept => "application/json"
         }
       }
       onos_hosts => {
         method => get
         user => "onos"
         password => "rocks"
         url => "http://localhost:8181/onos/v1/hosts"
         headers => {
           Accept => "application/json"
```

```
39          }
40        }
41        onos_links => {
42          method => get
43          user => "onos"
44          password => "rocks"
45          url => "http://localhost:8181/onos/v1/links"
46          headers => {
47            Accept => "application/json"
48          }
49        }
50        onos_packet_processors => {
51          method => get
52          user => "onos"
53          password => "rocks"
54          url => "http://localhost:8181/onos/v1/packet/processors"
55          headers => {
56            Accept => "application/json"
57          }
58        }
59        onos_statistics_ports => {
60          method => get
61          user => "onos"
62          password => "rocks"
63          url => "http://localhost:8181/onos/v1/statistics/ports"
64          headers => {
65            Accept => "application/json"
66          }
67        }
68        onos_system => {
69          method => get
70          user => "onos"
71          password => "rocks"
72          url => "http://localhost:8181/onos/v1/system"
73          headers => {
74            Accept => "application/json"
75          }
76        }
77      }
78      request_timeout => 60
79      # Supports "cron", "every", "at" and "in" schedules by rufus
    scheduler
80      schedule => { every => "5s"}
81      codec => "json"
82      # A hash of request metadata info (timing, response headers,
    etc.) will be sent here
```

```
83      metadata_target => "http_poller_metadata"
84      type => "onosapi"
85    }
86  }
87
88  filter {
89    split {
90      field => "applications"
91    }
92    split {
93      field => "devices"
94    }
95    split {
96      field => "flows"
97    }
98    split {
99      field => "[flows][selector][criteria]"
100   }
101   split {
102     field => "[flows][treatment][instructions]"
103   }
104   split {
105     field => "hosts"
106   }
107   split {
108     field => "links"
109   }
110   split {
111     field => "[links][src]"
112   }
113   split {
114     field => "[links][dst]"
115   }
116    split {
117     field => "packet-processors"
118   }
119   split {
120     field => "statistics"
121   }
122   split {
123     field => "[statistics][ports]"
124   }
125 }
126
127 output {
128   if [type] == "onosapi" {
```

```
129     elasticsearch {
130       hosts => ["localhost:9200"]
131       index => "onosapi-%{+YYYY.MM.dd}"
132     }
133   }
134 }
```

Listing D.2: Pipeline para el API REST de ONOS.

Lista de acrónimos

SDN *Software Defined Network.*

ELK *Elastic Logstash Kibana.*

ONOS *Open Network Operating System.*

IoT *Internet of Things.*

CLI *Command-line Interface.*

GUI *Graphical User Interface.*

IPFIX *Internet Protocol Flow Information Export.*

CPU *Central Processing Unit.*

RTT *Round-Trip Time.*

QoS *Quality of Service.*

WAN *Wide Area Network.*

IT *Information Technology.*

ICMP *Internet Control Message Protocol.*

TCP *Transmission Control Protocol.*

UDP *User Datagram Protocol.*

NFV *Network Function Virtualization.*

OSGi *Open Services Gateway initiative.*

AWS *Amazon Web Services.*

JSON *JavaScript Object Notation.*

SQL *Structured Query Language.*

JXM *Java Management Extensions.*

JDBC *Java Database Connectivity.*

SQS *Simple Queue Service.*

TSVB *Time Series Data Visualizer.*

MAC *Media Access Control.*

OSINT *Open Source INTelligence.*

Bibliografía

[1] Open Networking Foundation, "SDN architecture," 2014. [En línea]. Disponible en: https://www.elastic.co/guide/

[2] ——, "ONOS System Components," 2016. [En línea]. Disponible en: https://wiki.onosproject.org/display/ONOS/System+Components

[3] Prometheus, "Prometheus Overview," 2020. [En línea]. Disponible en: https://prometheus.io/docs/introduction/overview/

[4] Elastic, "Elastic Stack and Product Documentation," 2020. [En línea]. Disponible en: https://www.elastic.co/guide/

[5] W. Stallings, *Foundations of Modern Networking*, 1st ed. Pearson, 2016.

[6] C. Dueñas Santos, Y. A. Muro, and H. Cruz-Enriquez, "SDN Network vs. Traditional Network," 04 2017.

[7] Open Networking Foundation, "ONOS," 2020. [En línea]. Disponible en: https://wiki.onosproject.org/display/ONOS/ONOS

[8] Bazel, "Bazel overview," 2020. [En línea]. Disponible en: https://docs.bazel.build/versions/3.2.0/bazel-overview.html

[9] Mininet, "Mininet Overview," 2020. [En línea]. Disponible en: http://mininet.org/overview/

[10] Elastic, "Elasticsearch Introduction," 2020. [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html

[11] ——, "Logstash Introduction," 2020. [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/current/introduction.html

[12] ——, "Metricbeat overview," 2020. [En línea]. Disponible en: https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-overview.html

[13] ——, "Kibana — your window into the Elastic Stack," 2020. [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/introduction.html

[14] R. Cowart, "Network flow Monitoring (Netflow, sFlow and IPFIX) with the Elastic Stack," 2020. [En línea]. Disponible en: https://github.com/robcowart/elastiflow

[15] Grafana, "Grafana Features," 2020. [En línea]. Disponible en: https://grafana.com/grafana/

[16] Open vSwitch, "Production Quality, Multilayer Open Virtual Switch," 2014. [En línea]. Disponible en: https://www.openvswitch.org/

[17] B. Lantz, "OpenFlow."

[18] Cisco Systems, "Introduction to Cisco IOS NetFlow - A Technical Overview," 2012. [En línea]. Disponible en: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html

[19] Open Networking Foundation, "Open Network Operating System," 2020. [En línea]. Disponible en: https://github.com/opennetworkinglab/onos

[20] "Introduction to the Standard Directory Layout," 2020. [En línea]. Disponible en: https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html#introduction-to-the-standard-directory-layout

[21] "Java and Bazel," 2020. [En línea]. Disponible en: https://docs.bazel.build/versions/master/bazel-and-java.html#directory-structure

[22] Grafana, "Grafana," 2020. [En línea]. Disponible en: https://grafana.com/docs/

[23] Bazel, "Installing Bazel on Ubuntu," 2020. [En línea]. Disponible en: https://docs.bazel.build/versions/master/install-ubuntu.html

[24] ——, "Bazel," 2020. [En línea]. Disponible en: https://github.com/bazelbuild/bazel/releases

[25] Mininet, "Mininet Installation/Configuration Notes," 2020. [En línea]. Disponible en: https://github.com/mininet/mininet/blob/master/INSTALL

[26] Elastic, "Install Elasticsearch with Debian Package," 2020. [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/deb.html

[27] ——, “Installing Logstash,” 2020. [En línea]. Disponible en: https://www.elastic.co/guide/en/logstash/7.7/installing-logstash.html

[28] ——, “Install Metricbeat,” 2020. [En línea]. Disponible en: https://www.elastic.co/guide/en/beats/metricbeat/7.7/metricbeat-installation.html

[29] ——, “Installing Kibana,” 2020. [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/install.html

[30] Grafana, “Install Grafana on Debian or Ubuntu,” 2020. [En línea]. Disponible en: https://grafana.com/docs/grafana/latest/installation/debian/

[31] ——, “Download Grafana,” 2020. [En línea]. Disponible en: https://grafana.com/grafana/download

[32] ——, “Pie Chart plugin for Grafana,” 2020. [En línea]. Disponible en: https://grafana.com/grafana/plugins/grafana-piechart-panel?osource=grafana_plugin_list