UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN
INGENIERÍA DE TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DEVELOPMENT OF VISUALIZATION DASHBOARDS FOR REAL-TIME MONITORING ON SDN NETWORKS

DANIEL FERNÁNDEZ GORDO

2020

TRABAJO FIN DE GRADO

Título:	Desarrollo de dashboards de visualización para la monitorización en tiempo real de redes SDN.
Título (inglés):	Development of visualization dashboards for real-time monitoring on SDN networks
Autor:	Daniel Fernández Gordo
Tutor:	Álvaro Carrera Barroso
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:

Vocal:

Secretario:

Suplente:

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO FIN DE GRADO

DEVELOPMENT OF VISUALIZATION DASHBOARDS FOR REAL TIME MONITORING ON SDN NETWORKS

Daniel Fernández Gordo

Junio de 2020

Agradecimientos

Este trabajo ha sido posible gracias a todas esas personas que me han apoyado de diferentes maneras durante estos años de trayectoria.

Gracias a mis padres, que me han apoyado económicamente para que todo esto sea posible. A mis compañeros, por todas las experiencias compartidas fuera y dentro de la universidad. Sobre todo tengo que destacar a David y Diego, por todos esos buenos momentos, entre otras muchas personas de gran apego que han pasado inumerables horas a mi lado.

No puedo olvidar tampoco a mi tutor, Álvaro Carrera, y al Grupo de Sistemas Inteligentes, por los apoyos y el conocimiento que he obtenido durante la realización de este proyecto.

Gracias a todos.

Acknowledgement

This project has been achieved thanks to all those people who have supported me in different ways over the course of my undergraduate years of career.

Thanks to my parents, who have supported me financially to make all this possible. To my workmates and friends, for all the shared experiences shared outside and inside university. Above all, I have to recognize and thank David and Diego, for all those good times, among many other people of great value who have spent countless hours by my side.

I can not forget my tutor, Álvaro Carrera, and the Intelligent Systems Group, for the supports and knowledge I received over the course of this project.

Thanks to everyone.

Resumen

Factores como el aumento de la calidad de los servicios multimedia, junto con el aumento del número de terminales y su consumo, están generando la necesidad de transmitir más volumen de tráfico. Esto puede suponer en un futuro no muy lejano una sobrecarga de tráfico en las redes de telecomunicación. Será necesario, por lo tanto, una nueva forma de transmitir toda esta información de forma más eficiente.

Por está razón, actualmente, SDN (Software Defined Networking) está teniendo un gran impacto y está muy bien valorado en el campo de las redes, sobre todo debido a sus grandes ventajas frente a las redes convencionales. Una de sus principales ventajas es, por ejemplo, la separación de la capa de control de la capa de datos.

El sistema propuesto a continuación funciona como una herramienta de monitorización en tiempo real para redes SDN. Esta plataforma es especialmente útil para operadores de red que implanten una red basada en controlador SDN para el transporte de la información de sus abonados. Por lo tanto, surge también la necesidad de conocer el estado e información relevante de la red y sus equipos en tiempo real.

Se ha creado una simulación de una red SDN como modelo para generar tráfico, con la que será posible dar a entender el funcionamiento de la herramienta de monitorización, que siempre será posible transladarla a redes más complejas.

Como resultados pueden mostrarse, por ejemplo, interesantes gráficos sobre el tráfico que cursan los switches y sus interfaces, además de su disponibilidad, entre otras visualizaciones que serán representadas y agrupadas en dashboards.

Palabras clave: Software Defined Network (SDN), monitorización, estado de red, tiempo real.

Abstract

Factors, such as an increase in multimedia services,combined with a rise in the number of terminals and their data consumption are creating a need to transmit more traffic volume. This might mean, a traffic overload in the telecommunications networks in the not-too-distant future. Hence, it will be necessary to create a new way to transmit information more optimally to solve this possible future problem.

For this reason, SDN (Software Defined Networking) is having a huge impact and is greatly valued in the field of networks, due to its great advantages when compared to the operation of conventional networks, such as separating the control layer with the data layer.

The proposed project in the following document works as a real-time monitoring tool for SDN networks. This platform is especially useful for network operators that offer a service based on an SDN controller for the transport of their clients' information. For this reason, there will also be a need to monitor the network and its connected devices. As we use a monitoring platform we can know the status and relevant information in real time.

We have created a simulation of an SDN network as a model to generate traffic, with which it will be possible to understand the operation of the monitoring platform. We can also move it to complex networks in possible future purposes.

As shown by the results, interesting statistics about the switches and their interfaces will be shown. We can use it to check their availability and compare the traffic between interfaces, or to be informed about the different active flows transmitted between hosts. These are one of the visualizations, among others, which will be grouped and represented on dashboards.

Keywords: SDN, Monitoring, Network status, Real-time.

Contents

List of Figures

Glossary

API: Application Programming Interface
CLI: Command Line Interface
CSV: Comma Separated Values
DAG: Directed Acyclic Graph
GUI: Graphical User Interface
HTML: Hypertext Markup Language
HTTP: Hypertext Transfer Protocol
ISP: Internet Service Provider
JVM: Java Virtual Machine
MD-SAL: Model-Driven Service Abstraction Layer
MSO: Multiple Services Operator
NFV: Network Function Virtualisation
REST: Representational State Transfer
RTP: Real Time Transport Protocol
RTSP: Real Time Streaming Protocol
SDN: Software Defined Network
SMTP: Simple Mail Transfer Protocol
TCP: Transmission Control Protocol
UDP: User Datagram Protocol
VLC: VideoLAN Client
VM: Virtual Machine
YANG: Yet Another Next Generation

CHAPTER 1

Introduction

In this episode, in Section 1.1 we describe the motivations we had to make this project possible. Then, in the following Section, 1.2, we explain the aims of the project. Finally, in Section 1.3, we will depict the structure of the document.

1.1 Motivation

Telecommunication networks are always facing new challenges. The number of devices and the quality of service are increasing every year, therefore, this also increases the amount of traffic that is transmitted. According to experts [1], the number of new connected devices rises every year, and it is expected to surge to 125 Billion by 2030. One of the possible reasons is the implementation of 5G and IoT. The emerging IoT movement is impacting virtually all stages of industry and nearly all market areas. These devices jump 12 percent on average annually and global data transmissions are expected to increase from 20 to 25 percent annually to 50 percent per year, on average, in the next 15 years.

Additionally, an important role in this growth it is related to the expansion of Video-on-Demand services, like Netflix, and video streams, replacing commonly accepted TV broadcasting. All these facts indicate that there will be a huge amount of data in the

following years and we should be aware of the problems associated with this.

There are two main approaches to solving this issue. The first one could be improving link capacities, and we are trying to move from copper pair to optical fiber because of its better capacities and less attenuation, by which transport of the information is improved. However, even if this is improved there will always be physical limitations we can not overcome.

The second one, is the implementation of SDN networks, on which the computer networking industry is focusing. This new standard allows for more flexibility when it comes to managing the network. We can relive network elements depending on the current needs. In conventional networks, changing traffic policies is more laborious and comes with higher cost: each modification have to be individually injected. Besides, there is always a change of possible incompatibility with other policies.

Software Defined Networks features have a big advantage over conventional ones. They are based on a controller that plays the brain of the network, in which we can manage the elements connected. Besides, one of the most remarkable characteristics is the separation between the control layer and the data layer. Each node of the network is connected to the controller and their communication with each other is possible thanks to Openflow protocol, which is specially designed for this purpose and is becoming standard protocol for the communication between the different elements of the system on SDN networks.

This superiority is being noted by the communication networks community. According to Cisco, one of the most important manufacturers in the field of networking, SDN technology is considered to be the new generation networking paradigm because of its big advantages like coordination and speed. Since policy modifications can now be easily and quickly introduced, less time and effort is being spent on network management. As a result, it's expected to be a new way Telecom Companies operate their networks and this field will be transformed in a few years.

For this reason, we will implement a monitoring tool for SDN networks, which that can be of great use to network operators. They can apply it to their infrastructures and be aware of the real-time status of their links and devices. This platform will be implemented and designed to work on SDN networks, so there needs to be a network controller from which the information can be extracted. Network operators will be able to use it to diagnose possible problems, such as detecting the saturation of certain interfaces, among other applications.

1.2 Project Goals

For the purpose of developing the system mentioned in the previous section, the following objectives are defined:

1. To define and develop a simulated small-scale SDN network with traffic between hosts.
2. To collect information from the controller and the sflow collector in order to get the real-time status of the network devices and traffic.
3. To make use of this collected data to create visualization dashboards.
4. To reproduce use cases to review the good functioning of the tool for monitoring the system.

1.3 Structure of the Document

The document is structured in 6 chapters.

In Chapter 1, Introduction, we make a general presentation and we announce the aim of the project. Then, in Chapter 2, Enabling Technologies, we depict the tools and technologies used to implement this project. To continue, in Chapter 3, Architecture, we describe the proposed architecture explaining their elements. Next, in Chapter 4, Prototype, we define the prototype considering the architecture explained previously. Then, in Chapter 5, Use Cases and Results, we represent use cases to check the good functioning of the system for a network operator. To finish, in Chapter 6, Conclusion, we show some conclusions and possible future work.

CHAPTER 2

Enabling Technologies

To conceive this project defined below, we have taken advantage of several technologies of which some have already been used in previous work by members of our research community. The entirety of the project is embedded in Docker Containers, which facilities the possibility of moving the project from one host to another, just using the same Docker Images. Definitely, with the use of Docker we add a grade of sophistication, but we obtain more flexibility and adaptability. As a way to develop this proposed system we applied technologies to implement a SDN network, such as Open vSwitch and Openflow. In addition, we use technologies to collect, store and display information from the network.

2.1 Introduction

In this second chapter of the document, in Section 2.2, first we describe the components of the SDN networks, such as the switches, protocols and controller. Then, in Section 2.3 we explain the deployment environment in which the project is embedded. To finish this Chapter, in Section 2.4 we illustrate the Big Data technologies used to collect, store and represent the information from the network.

2.2 Software Defined Networking

In this section, we describe the technologies used in the implementation of the SDN used in the proposed system. First, we explain the main characteristics of these networks in Section 2.2.1. Then, in Section 2.2.2, we define the SDN Architecture. Next, we offer an explanation of the protocol used in the communications between controller and nodes in Section 2.2.3. Then, we define the virtual switches used as nodes in our system in Section 2.2.4. To finish, in Section 2.2.5, we go into detail about the different SDN controllers.

2.2.1 SDN paradigm and main characteristics

SDN has introduced a new paradigm in network computing for data network management, where from a central controller we can manage the operation and configuration of network devices. While in conventional networking each network element has its own control plane, which is accessible through vendor-specific channels, in SDN the network elements lose their "intelligence" in favor of a network controller which holds all the intelligence and the data needed for the correct functioning of the network. This paradigm has become one of the most important innovations in the computer networking field in recent years.

Software defined networks separate the control layer, which manages the network, from the data layer where the traffic flows. Therefore, the management of the network can be programmed because forwarding and routing uses a different plane.

Moreover, network elements can be activated and deactivated when needed dynamically, depending on the specific use. These features increase the agility and the ease of administration, which is completely centralized. SDN simplifies the operation and design of the network because of this centralization, instead of giving instructions to several devices with diverse protocols we focus on the controller: This network administrator has a global view of the status of the network, so it is easier to take decisions, being aware of the services provided within it.

To conclude, we can say that SDN networks provide several advantages in different practices: centralization, agility, dynamic activation of elements, ease of management and design, and global view of the status of the network.

2.2.2 SDN Architecture

The Figure 2.1, represented below serves as an example of the SDN architecture.

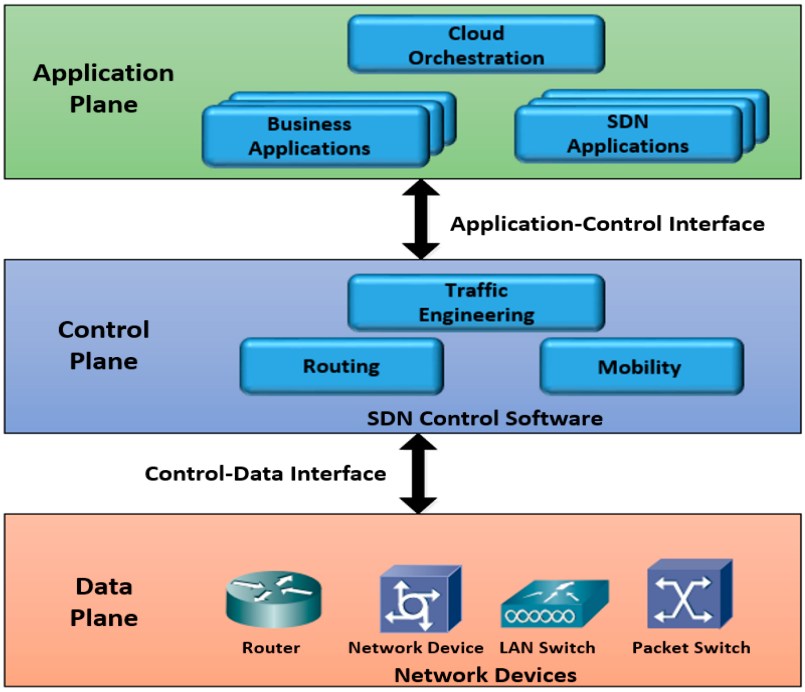

Figure 2.1: SDN architecture

1. **The Application plane** is open area to develop as much innovative application as possible by leveraging all the network information about network topology, network state or network statistics. There can be different types of applications, such as networking configuration and management applications, network monitoring, network troubleshooting, network policies and security, among others. These applications can provide various solutions for data center networks for example. In the particular case of this project we will be developing a monitoring application.
2. **The Control plane** is where the intelligence and logic of the network reside, and where the control network infrastructure is defined. Here in this plane, a lot of business logic is being written in controller to fetch and maintain different types of network information, state details, topology details, statistics details, and more.
3. **The Infrastructure plane** is composed of various networking equipment, like routers, switches and other network devices. It could be a set of network switches and routers in the data center. This layer would be the physical one over which network vitalization would be installed through the control layer (where SDN controllers would be placed and manage the physical network).

2.2.3 OpenFlow

OpenFlow is considered the standardized protocol for SDN networks. It allows for the communication between nodes and the controller itself, completely necessary for the good functioning for a SDN network. This protocol works in the infrastructure layer, so it manages the forwarding plane of an SDN switch, and its use allows us to move the network control of the switches to the SDN controller software. OpenFlow implements packet routing, the most basic service in networking.

OpenFlow permits the network to be programmed based on flow traffic and is adapted to the structure of traffic policies, which make possible the response to changes in real time. A flow defines a set of matching rules and actions. This means that each incoming packet is evaluated in order to match into a flow entry. The action indicated in the rule is performed when a flow is matched with a table entry from the table.

Every switch has its own flow tables stored inside. Openflow protocol manages the way in which the controller manages each flow table, for example by creating or deleting flow entries, or by defining how long they live without being used. An SDN flow table is composed of 3 main parts:

1. **Fields**: MAC source, MAC destination, IP source,IP destination, Destination port, VLAN, etc.
2. **Action**: Behaviour of the traffic flow (drop, send it to controller, send it by a certain port...)
3. **Data traffic statistics**: Statistics as a counter that have to coincide with the rule.

Flow table	Switch Port	MAC src	MAC dst	Ether type	VLAN ID	Src IP	Dst IP	Proto No.	TCP S_port	TCP D_port	Action	Counter
Switching	*	*	00:1f..	*	*	*	*	*	*	*	Port1	243
Flow Switching	Port3	00:20..	00:2f..	0800	vlan1	1.2.3.4	1.2.3.9	4	4666	80	Port7	123
Routing	*	*	*	*	*	*	1.2.3.4	*	*	*	Port6	452
VLAN Switching	*	*	00:3f..	*	vlan2	*	*	*	*	*	Port6 Port7 Port8	2341
Firewall	*	*	*	*	*	*	*	*	*	22	Drop	544
Default Route	*	*	*	*	*	*	*	*	*	*	Port1	1364

Figure 2.2: Flow table example

An example is defined in Figure 2.2, where we can look at the different fields of a flow table and its action for each flow. If the field is filled with an * it means that any value is allowed.

2.2.4 Open vSwitch

Open vSwitch is an open source virtual multilayer switch that supports the Openflow protocol, described in Section 2.2.3. It is commonly used in SDN environments and it fits perfectly with our needs because it is designed to be controlled by an SDN controller. Open vSwitch defines and deploys the nodes which will be acting as switches to interconnect virtual hosts in our virtualized network. Whenever we mention "switches", it must be understood that we are not referring to the traditional concept of switch, a level-two routing element, but to an element that implements rules on the 7 OSI levels entirely [2].

Open vSwitch can operate both as a software-based network switch running within a virtual machine, and as the control stack for dedicated switching hardware. In our case we use it in a virtualized approach.

Open Vswitch supports the following features: L2-L4 matching, VLANs with trunking, Tunneling protocols such as GRE, Remote configuration protocol, Multitable forwarding pipeline, Monitoring via sFlow, Spanning Tree Protocol, OpenFlow protocol.

In Figure 2.3 is represented the architecture of a Open vSwitch.

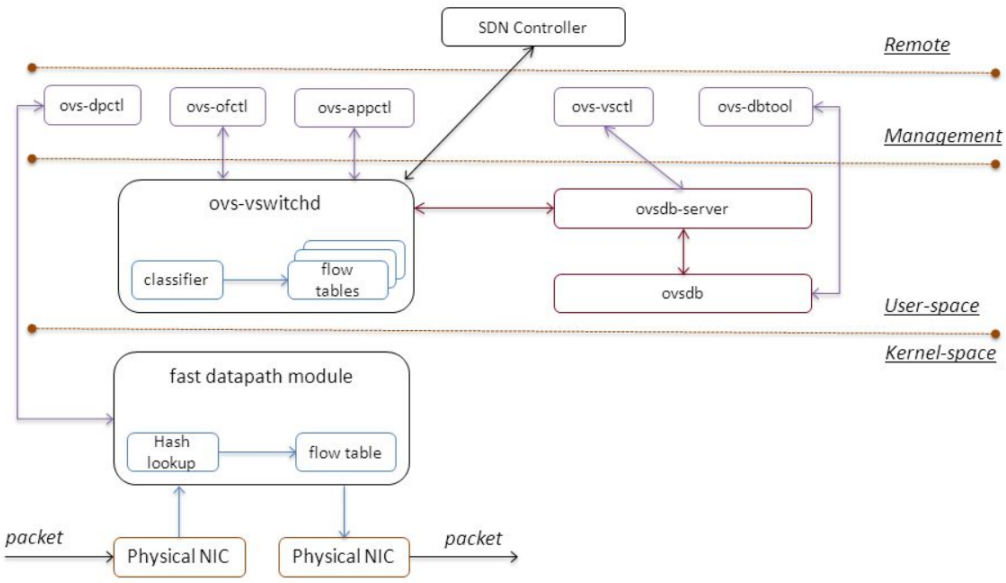

Figure 2.3: Open vSwitch Architecture

Considering Figure 2.3, the daemon *ovs-vswitchd* is built with the view to control every Open vSwitch switch defined in the local machine. It communicates with *ovsdb-server*,the Open vSwitch database server, that provides interfaces to multiple databases *ovsdb*, where there are stored in several configuration variables, for example Flow tables. Each switch has to communicate with the SDN controller by an interface connected in the machine that hosts the SDN controller.

2.2.5 SDN Controller

An SDN Controller is the application that acts as a strategic control point in a software-defined network. Essentially, it is the "brains" of the network. An SDN controller manages flow control to the switches,via southbound APIs, and the applications and business logic 'above', like monitoring tools, via northbound APIs, to deploy intelligent networks. Just like the other elements of a SDN network, SDN controller uses OpenFlow protocol to communicate with switches.

Some of the basic tasks of a controller is to show every device connected within the network and the capabilities of each, gathering network statistics, and other monitoring functions. Extensions can always be inserted that enhance the functionality and support more advanced capabilities.

Currently there are many commercial and open-source controllers available, some of them are: Nox, Pox, Cisco APIC, NSX, Floodlight, ONOS, Opendaylight, Ryu, to name a few.

The first SDN controller to be created was NOX, but we go into detail in the most important the most and used: Opendaylight, ONOS and Floodlight.

Opendaylight: It is the open-source SDN controller that we use in this project. It is the most complex and is inter operable with several different proprietary applications and provides a flexible common platform underpinning a wide variety of use Cases. This controller is able to manage networks of different sizes and we can integrate it with the OpenStack due to its REST API.

ONOS: (Open Network Operating System) is an open-source controller that operates in the control plane of a SDN network.The most important benefit of an operating system is that it provides a useful and usable platform for software programs designed for a particular application or use case. ONOS applications and use cases often consist of customized communication routing, management, or monitoring services for software-defined networks. The ONOS kernel and core services, as well as ONOS applications, are written in Java

Floodlight: is an SDN Controller developed by an open community of developers, many of whom are from Big Switch Networks, which uses with the OpenFlow protocol to orchestrate traffic flows in a software-defined networking (SDN) environment

2.3 Deployment Environment

This project is deployed inside a simulation environment, which allows us to reproduce the same characteristics and behavior in a small-scale sample. We can always move the project to a bigger scenario after debugging and detecting possible errors that could affect the behavior of the system.

2.3.1 Docker

Docker is the main environment in which the project is deployed. It is an open-source tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow us to deploy each service in a different container.

We can say that it is a bit like a virtual machine, but unlike a virtual machine, it is not created by a virtual operating system. Docker uses the same Linux Kernel as the system that they are running on. This gives a significant performance boost and reduces the size of the application. It is used as a lightweight virtualization mechanism at the operating system level.

This service packaging is done through Docker images that contain the necessary information to create a container with services inside. This means that we can deploy the same environment in different hosts just using the same images. For this reason, Docker is very useful and provides flexibility to our system.

These are the main components needed to understand the Docker Architecture:

- **Dockerfile**: Is a document where we define the content of a Docker image. Here we find here the instructions needed to build this image, which are executed in sequence as commands.
- **Image**: Is a file built to create a Docker container. We can create a local image by using a Dockerfile, or downloading it from a repository stored on the Cloud.
- **Container**: Is an executable instance of a Docker image. A Docker container is the virtualized environment used to provide a specific service.

- **Docker compose**: Is a tool in which we execute a YAML file to run different containers in the same network.

The picture bellow is a diagram of the Docker architecture, with the elements described before.

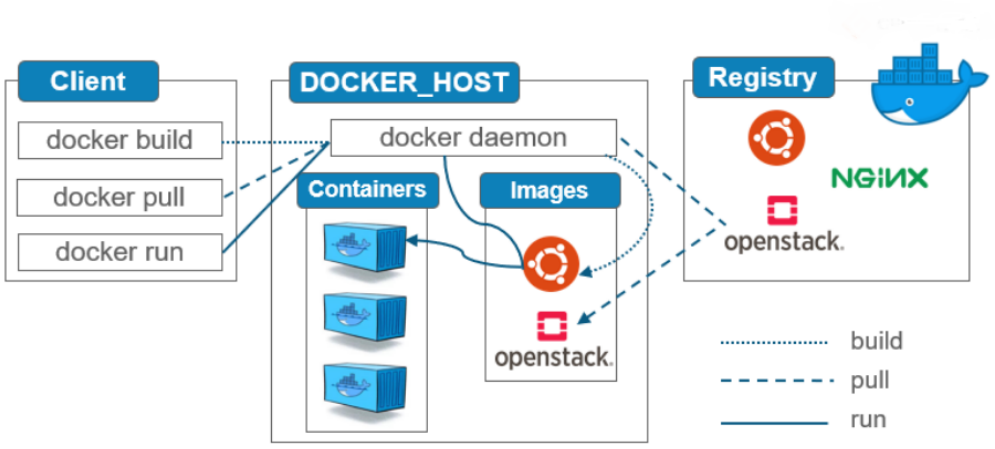

Figure 2.4: Docker Architecture

2.3.2 Mininet

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow protocol, described in Section 2.2.3.

Mininet provides an easy way to achieve correct system behavior and run real code including standard Linux network applications as well as the real Linux kernel. Because of this, the code developed and tested on Mininet, for an OpenFlow controller, modified switch, or host, can move to a real system with minimal changes, for real-world testing.

This network emulator allows us to create, customize and share SDN networks prototype while providing the adaptability to migrate to hardware.

Mininet provides an inexpensive network testbed for developing OpenFlow applications, an extensible Python API for network creation and experimentation, multiple concurrent developers with the possibility if working independently on the same topology, complex topology testing, CLI for debugging and an easy use out of the box without programming.

The following figure describe perfectly what Mininet is:

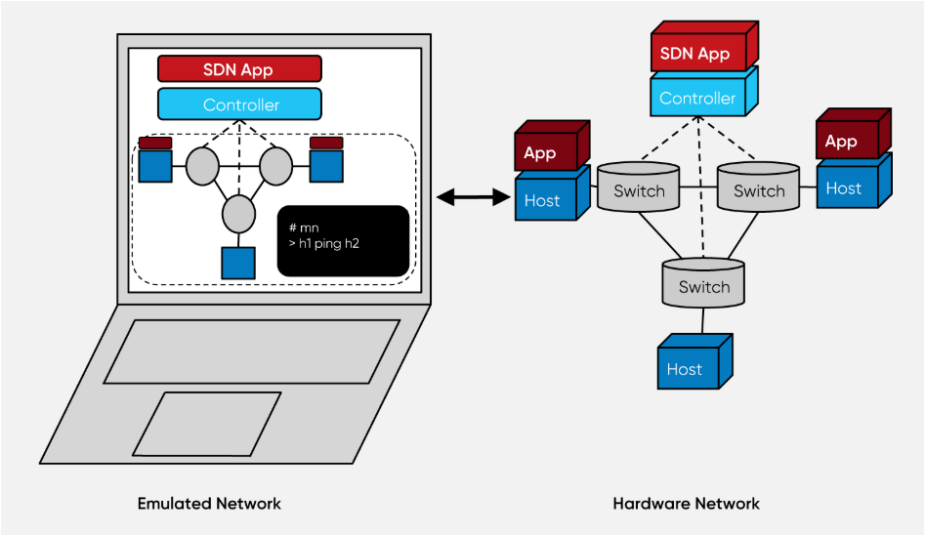

Figure 2.5: Mininet: Emulated network

For this reason, Mininet is our deployment environment to emulate a network. It provides the same behaviour as a hardware network, but in an emulated environment, which we will be using inside a Docker Container.

2.4 Big Data Technologies

As Big Data Technology we make use of the **Elastic Stack (ELK)**: Logstash , Elasticsearch and Kibana. Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, Elasticsearch is a search and analytics engine and Kibana visualize data with charts and graphs. This combination gives us the mechanism to collect, store and visualize data coming from the SDN controller and the sflow collector.

Figure 2.6: Elastic Stack

2.4.1 Logstash

Logstash is a powerful open-source data collector with real-time pipelining capabilities. We use Logstash as a tool to collect, process and forward events and log messages.

Logstash data collection is accomplished via configurable input plugins. Each application has its own format to display these events. However, Logstash has the capacity to adapt its input, with the use of different plugins, to reach and use the source of information in different formats.

Once an input plugin has collected data, it can be processed by any number of filters which modify the event data.

Finally, logstash make use of output plugins which can send the filtered events to external programs, including of course, Elasticsearch.

As we said before, this tool is based on pipelines, that combine different kind of plugins for inputs, filters and outputs:

- **Inputs**: An input plugin enables a specific source of events to be read by Logstash. We have at our disposal many different plugins for different uses, some of the most used are the following:
  - *Http Poller*: Decodes the output of an HTTP API into events. We can set when to periodically poll from the urls.
  - *Beats*: Enables Logstash to receive events from the Elastic Beats framework.
  - *Syslog*: Read syslog messages as events over the network.
  - *Heartbeat*: Generates heartbeat events for testing.
  - *File*: Stream events from files.
- **Filters**: A filter performs intermediary processing on an event. Filters are often applied conditionally depending on the characteristics of the event. Some of the most used are:
  - *Mutate*: Performs mutations on fields, for example to rename, replace, merge, etc.
  - *Grok*: Parses unstructured event data into fields.
  - *Drop*: Skip the event.
  - *XML*: Parses XML into fields.

  - *Split*: Splits multi-line messages into distinct events

- **Outputs**: An output plugin sends event data to a particular destination. Outputs are the final stage in the event pipeline.

  In our use case we use the output to send the data to Elasticsearch.

2.4.2 Elasticsearch

Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows us to store, search, and analyze big volumes of data quickly and in near real time. It is generally used as a technology that powers applications which have complex search features and requirements. It works with JSON records.

To better understand Elasticsearch and its usage it is good to have a general understanding of the main backend components:

- **Node**: A node is a single server that is part of a cluster, which stores our data, and participates in the cluster's indexing and search capabilities. They are identified by a name which is random by default and is assigned at startup.

- **Cluster**: A cluster is a collection of one or more nodes that together holds your entire data and provides federated indexing and search capabilities.

- **Index**: The index is a collection of documents that have similar characteristics. For example, we can have an index for topology information, another for data flows...etc. An index is identified by a unique name and we can define as many indexes as we want to classify our data.

- **Document**: A document is a basic unit of information that can be indexed. This document is expressed in JSON. Within an index, you can store as many documents as you need.

- **Shard and Replicas**: Elasticsearch provides the ability to subdivide your index into multiple pieces called shards. When you create an index, you can simply define the number of shards that you want. Each shard is in itself a fully-functional and independent "index" that can be hosted on any node in the cluster.

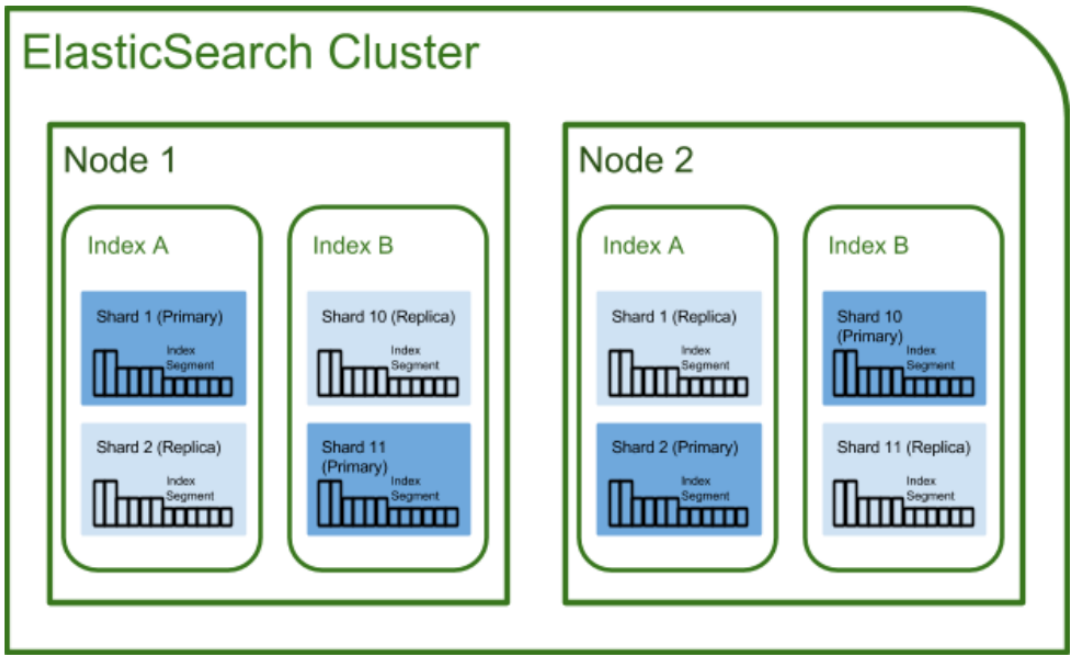

Figure 2.7: Elasticsearch Arquitecture

Elasticsearch has an extra packet, X-Pack, which adds additional features. It is mandatory to provide a valid license to use these extra characteristics. There is also a 30 day trial available to make use of it.

2.4.3 Kibana

Kibana is an open source frontend application that sits on top of the Elastic Stack, providing search and data visualization capabilities for data indexed in Elasticsearch. Kibana also acts as the user interface for monitoring, managing, and securing an Elastic Stack cluster.

We use Kibana for searching, viewing, and visualizing data indexed in Elasticsearch and analyzing the data through the creation of bar charts, pie charts, tables, histograms, and maps. A dashboard view combines these visual elements to be shared to provide real-time analytical views of large data volumes.

Kibana offers its visualization through a web interface on port TCP 5601. However, before starting to use Kibana, we must configure Elasticsearch and define an index pattern.

To understand how it works, these are the main features to explore and create dashboards :

- **Discover**: Discover enables us to explore data with Kibana's data discovery functions. There is access to every document in every index that matches the selected index pattern. It can submit search queries, filter the search results, and view document data. Besides, it is possible to see the number of documents that match the search query and get field value statistics.
- **Time window**: In kibana we can set a time window to view a specific data or to see a large image of the information.
- **Console interface**: It allows the use of Developer Tools console (Dev Tools), in which you can compose request to send to Elasticsearch and view their responses.
- **Visualizations**: There are different type of visualizations available to display our data: Bar charts, Pie charts, tables, histograms...
- **Dashboards**: A dashboard is a collection of visualizations, typically in real-time. Dashboards enable us to drill down into details and provide a quick view of visualizations.

In the Figure 2.8, represented below, we can see an example of a Kibana Dashboard.

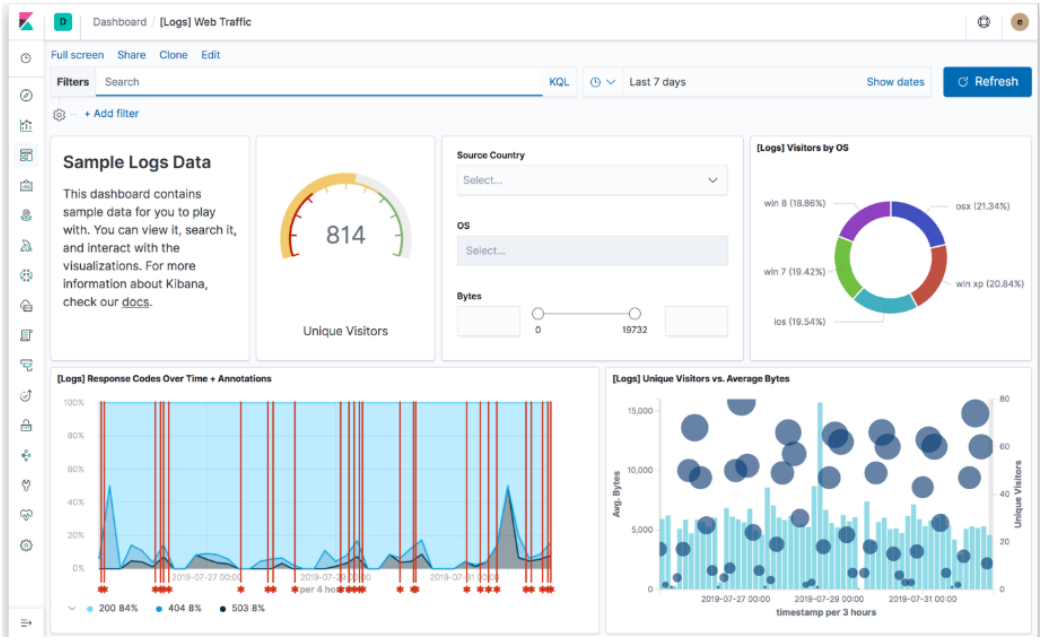

Figure 2.8: Kibana Dashboard Example

As we can see, there are several visualizations grouped in a dashboarh. Therefore, it is easy for the end user to look at all the information represented inside it.

CHAPTER 3

Architecture

In this chapter, we describe the architecture proposed in this project, including the design phase and implementation. We have designed a system for monitoring an SDN network. In order to perform it, we first need an SDN network and then, a system to collect information from it.

First, we depict the modules that compose the proposed architecture, emphasizing the tasks implemented by each one. Then we describe the connection of these subsystems, obtaining the complete architecture.

3.1 Introduction

In the first place, in Section 3.2, the architecture of the system developed in this document is presented. Next, in Section 3.3 we describe the base of the project: an SDN network. Then, in Section 3.4 we represent how the data is ingested in the collection system. Next, in Section 3.5 an overview about the data processing subsystem and the tools involved are described. To continue, in Section 3.6 we explain how data persists in the system. Finally, in Section 3.7 we make clear how to represent this data in the visualization module.

3.2 Architecture Overview

The general architecture of the system is represented in Figure 3.1. As we can see, the project is embedded in a Docker environment and it is made of 5 layers that we explain in this chapter of the document: an infrastructure layer, ingestion layer, processing layer, persistence layer and the visualization module to display the processed data.

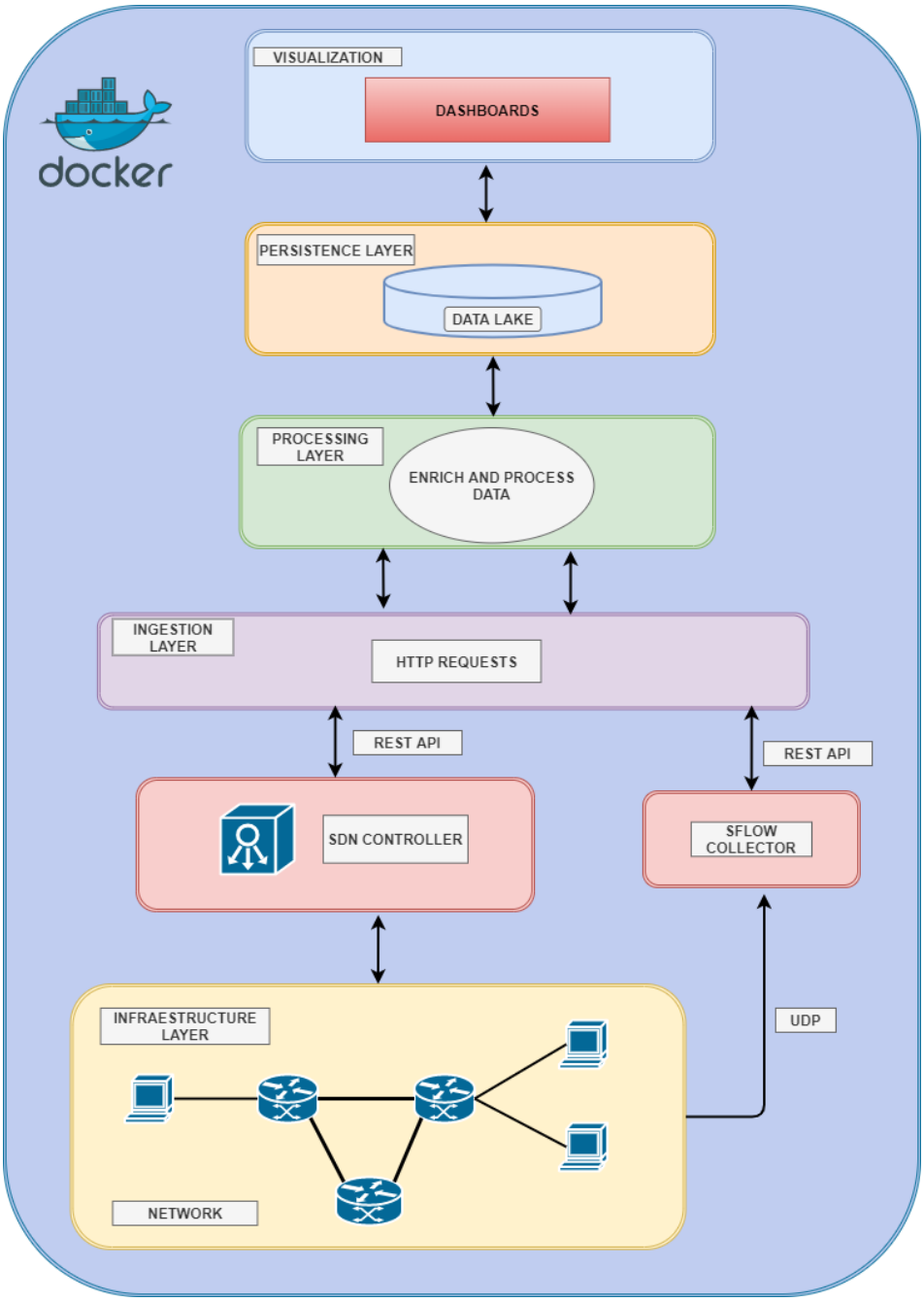

Figure 3.1: Overview of the general architecture.

At the bottom of the architecture we have the infrastructure layer, in which we deploy a network and generate traffic on it. As indicated in the figure above, the network is controlled by an external SDN controller, and not the controller of the SDN network by default. In addition, we have to use a sflow collector as an element to extract traffic information that is not provided by the controller. The controller holds only information of the devices and links of the network, but it does not have much information about traffic.

Then, we create a set of modules to collect, process and store information from the controller and from the sflow collector.

Finally, we represent all the information in the visualization module, which we use as the monitoring tool.

3.3 Infrastructure layer: SDN network

The infrastructure layer is the base on which the project is mounted. We need to create first a scenario in which we can simulate real behaviour. To obtain these results we have to use a network emulator because we don't have the means and the capacity to create a real one.

SDN network emulators provide their own SDN controller. In this project we use an external SDN controller, in particular we make use of Opendaylight controller, listed and detailed in Section 2.2.5.

It is one of the most used and with better characteristics in this field. In addition, it is the largest open-source controller and a platform for customizing and automating networks of any size and scale. Opendaylight controller has the intelligence to manage complex networks and to be efficient against changes to the network.

To extract the information from it, we use the API REST available. We can make queries and get the real time status of the links and the devices connected. Just in case it needs to be used a different SDN controller, almost everyone has an available REST API to extract information from it.

In Figure 3.2, we can see an example of a network created with an emulator, being C0 the default controller.

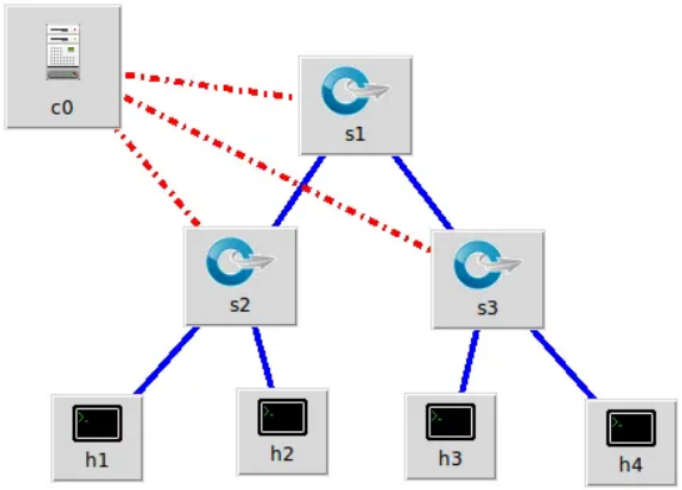

Figure 3.2: SDN network created with network emulator.

3.4 Data Ingestion

The data ingestion layer is the plane in which we carry out the process of obtaining and importing data for immediate use or storage in a database. This data is streamed in real time, so data items are imported at periodic intervals of time.

Our goal is to extract information from the designed network. There are different methods to implement this action. The first one is to extract data and network status directly from the devices, such as switches or hosts machine, for example by obtaining information and monitoring their interfaces directly. However, it is more arduous and we do not take advantage of the SDN network. In addition, we would need higher computing resources depending on the number network elements. This is the method we would use it if we had a conventional network, but we have an SDN network instead.

This is why our method of extracting data is by taking advantage of the SDN controller. It is not designed in the first place to be used as a machine to get information, but it is one of their utilities via its Northbound API. However, due to controller limitations, we have to extract traffic information from the switches of the network. The SDN controller does not provide much information about flows and traffic, and this data could be very useful if we add it to our visualization dashboards.

For this reason, we need to extract the main information from the controller, such as the network and devices status, and also collect traffic data from switches via the sflow collector. We need this hybrid approach to make our monitoring tool more complex and

complete.

The method for extracting data from the SDN controller is by querying its Northbound interface periodically, in which we can make use of the REST API. This Northbound interface makes the connection possible and provides integration between the controller and services or applications running over it. The procedure to obtain this information is based on making specific HTTP requests via URLs to receive network information represented in JSON format.

On the other hand, we need to extract not only devices and link status, but traffic information. This process can be done in many ways because there are different systems that can work as an sflow collector, for example Logstash and some SDN collector, such as Opendaylight. However, it needs an external data base and this makes this process more complex and less optimal to obtain the same results. For this reason, we decided to use a specific application designed to collect and process sflow data. This application also has a REST API which we can use to make queries and extract this processed data.

Summing up, we build the ingestion process using 2 mechanisms to obtain a more complex data set. As we can see in Figure 3.1, we have two sources of data ingestion in the system architecture. The first one collects data directly from the SDN controller through its Northbound API, while the second one makes use of network nodes, which work as sflow agents, to collect traffic information and send it to an sflow collector.

3.5 Data Processing

In this section we describe the data processing module, in which we transform and enrich the information provided by the Data Ingestion Module, explained in Section 3.4.

Once we have the method to obtain real-time data by querying the SDN controller and the sflow collector periodically, we have to treat and prepare this raw information provided in JSON format. However, raw data usually tends to be noisy, and it usually has useless and non-relevant information.

In order to do this, we can use Big Data software platforms, such as Logstash and Elasticsearch. Using Logstash we can make use of several filters to extract only the fields we need to build the monitoring tool. In addition, it can be used to index and classify high volumes of collected data in order to simplify further classifications.

The goal of this activity is to make sure that end users can query for events and correlations between fields, for example, a Switch and the its interfaces. Since the data is in

JSON format, we have to be aware of the fields disposed in "tree" format and which one contains each other in order to build this relations between fields.

We only extract fields regarding the general status of the network, we discard the rest. As we have previously stated, this data allows us to obtain a wide view of the current status of the network. Depending on the use case, we can focus on specific areas of the network, such as traffic statistics or link status.

3.6 Data persistence

The data persistence module is used to store the received information provided from previous processes and modules. It is a central point where we can retrieve stored data when required. This data can be used by the visualization module to build graphs and other kinds of visualizations.

The visualization module interacts directly with the database. There is not any user interface or the user involved with database interaction. Therefore, the end user of the system can not make queries to this data store. In this case, the visualization module interacts with the database and retrieves all required data by making specific queries, as we can see in Figure 3.3.

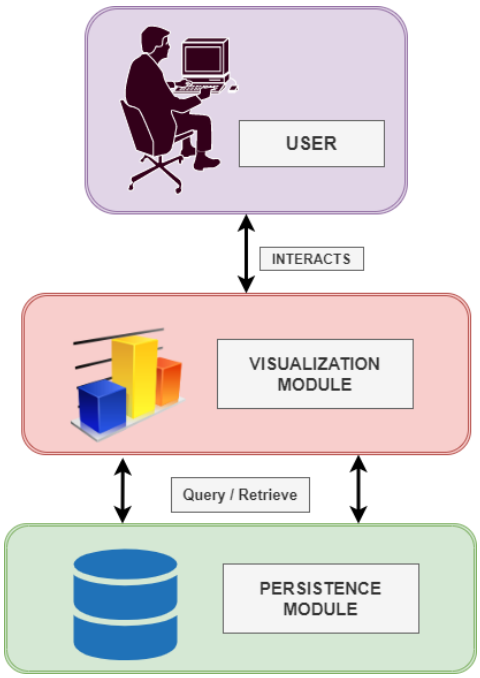

Figure 3.3: Data persistence integration.

This data persistence module provides an implementation of a repository in which it is allowed to save, delete, find and search for objects stored.

We can make use of different systems that can work as a database, such as Elasticsearch, described in Section 2.4.2, and which we use in this architecture playing the role of Data Lake. However, we could use any other kind of database, but we take advantage of the **Elastic Stack**. The only thing we have to keep in mind if we use Elasticsearch 2.4.2, is that it has a different nomenclature compared to a relational database:

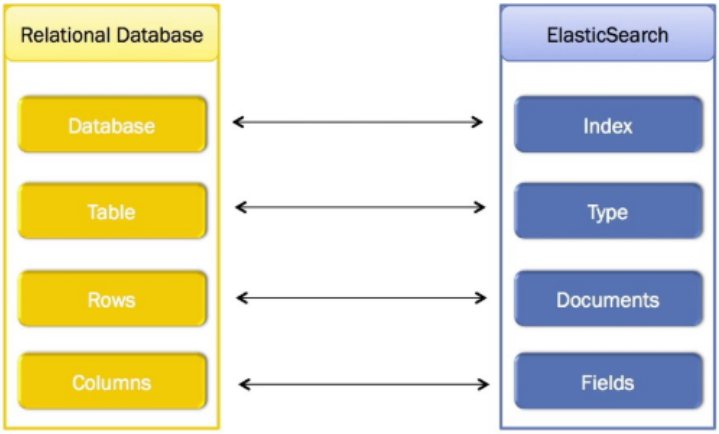

Figure 3.4: Elasticsearch and Relational database nomenclature.

3.7 Visualization

The data visualization module is basically a graphical representation of the processed information stored in the database. It has a visual content through which the end user can understand the significance of data, since this data is not readable by humans. Neither does it gives a good awareness of the network usage. Therefore, we consider that a visualization module is needed in order to greatly improve the usability of the proposed system.

This module allows us to analyze a massive amount of information using visual elements, such as graphs and tables. There are different types of visualizations that can help to understand the behaviour of the network. Each type has a specific purpose, depending on how we want to represent the data source.

The introduced monitoring system presents a user friendly interface and innovative approach for network monitoring. This tool is important to examine the network status and make it easier for the final user, so we can identify areas that need improvement attention.

As we explained before, represented in Figure 3.3, the unique data source of this module is the database. The visualization subsystem collects data from this database and applies graphic type to the data. Generally, the way we have to represent this information is by dashboards, a visualization collection.

One of the possibilities considered in order to build this last element of the architecture is the use of the Elastic Stack. This combination, mentioned in Section 2.4, is an appropriate solution to implement and reach the results we need.

Kibana is the visualization tool provided by the Elastic Stack. It has a Web interface in which we can build dashboards with the data stored in Elasticsearch. Since both are elements of the Stack, they are specifically designed to work together. We can search, view and interact with data stored in Elasticsearch indices and perform advanced data analysis to visualize data in a variety of charts and tables.

CHAPTER 4

Prototype

Once we have depicted the architecture in Chapter 3, we describe the implementation of such architecture. We developed a network simulation in which we generate traffic between hosts. Then, we implemented a data collector that extracts data from the SDN network controller and the sflow collector, and stores the collected information in the Data Lake, which was also implemented. Finally, we implemented the Visualization module to work as a monitoring tool.

4.1 Introduction

In this chapter, we detail the implementation of the prototype for our monitoring tool. First, we focus on the architecture of the prototype implemented in Section 4.2. Then, in Section 4.3, we describe the design and implementation of the simulated network. Next, in Section 4.4, we explain the selected SDN controller used in this project. Then, we describe the specific data ingestion elements in Section 4.5. Next, in Section 4.6 we explain how we process the data, which we store in our persistence module explained in Section 4.7. Finally, in Section 4.8 we go into detail about the visualization module to build our monitoring tool.

4.2 Architectural development

We have developed the architecture as we can see in the following full scheme in Fig. 4.1.

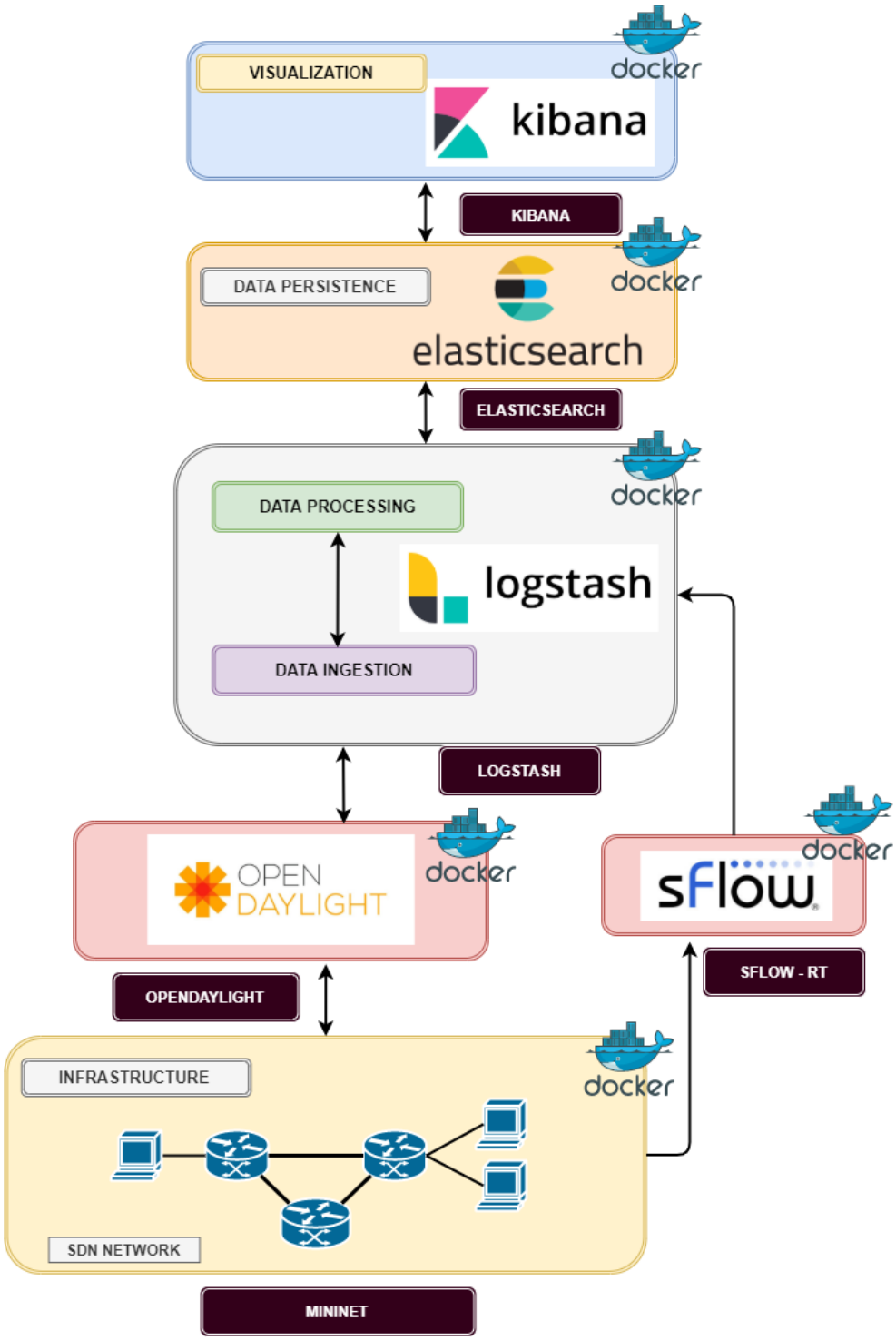

Figure 4.1: Overview of the prototype architecture using Docker containers.

We have used Docker in the architecture deployment, due to its ability to provide flexibility and isolation of the environment. Each module is running, without the hardware requirements of a Virtual Machine (VM). In addition, we have used Docker-Compose to build the project in the same docker network, in which we have six Docker containers running simultaneously as we can see in the figure above.

We have focused on using the network controller as the data source to get information about the status of the network, and we have chosen Opendaylight as the network controller for this task. In addition, we use SFLOW-RT as the sflow collector to extract traffic and flow information.

We use the Elastic Stack, previously mentioned in Section 2.4, to extract, process, store and visualize the information of the network. The ingestion and processing task is done with Logstash. The Data Lake has been implemented using Elasticsearch in order to allow for a flexible indexing and storing of the data. Finally, Kibana has the function to represent all this information in dashboards.

4.3 Infrastructure layer

In this section we describe the implementation of the network simulation. We use Mininet as the tool to emulate and create our network topology. As we explained in Section 2.3.2, we rely onMininet because it is widely used for these purposes.

For our particular scenario, we build a network with 5 switches and 5 hosts as a small-scale telecommunication network representation. This scenario allows us to reproduce real behaviour in which we can generate traffic between nodes. Each host has its own IP address and the IP forwarding on the switch is available to make the connection possible between them. These hosts will act as clients or servers in our network.

Mininet library "Topo" was the base used to create a Python script in which we specify the number of switches and hosts, and how they connect with each other. We execute the following command to build the network:

```
mn –custom topo.py –topo mytopo –controller=remote,ip=10.0.1.12,port=6633
```

When we execute this command, we specify this python script to create the predefined custom network. We also indicate the IP and port of the external network controller. This IP target is the IP of the Docker container where we have the Opendaylight controller.

Here we can see the model of the network created in this project:

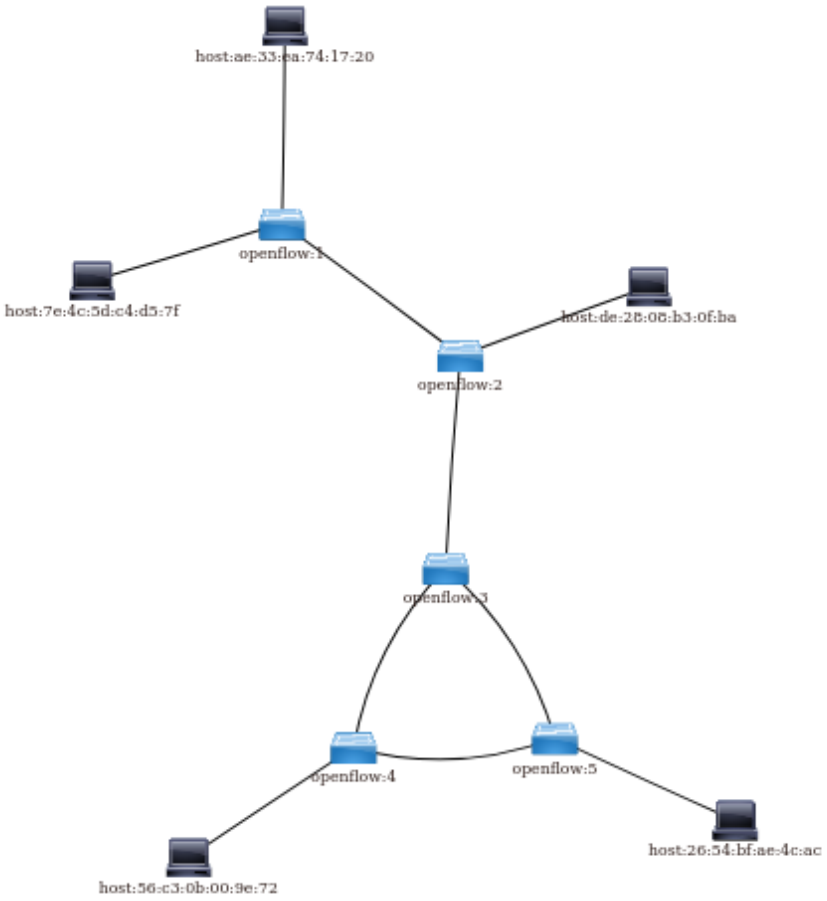

Figure 4.2: Topology prototype.

The switches have a unique id and they are named with the name of the SDN protocol, Openflow, followed by a numeric value starting from 1. For example, we can see that the first one is Openflow:1.

Once we have the network mounted, we have to create traffic between hosts. For it, we make use of Iperf as the tool to create a volume traffic between them. Fundamentally, we have to indicate a host which works as a server, and another host that works as a client.

4.4 Control layer: SDN Controller

As we described in Section 2.2.1, the SDN paradigm requires the presence of a network controller, which is considered the *brain* that takes over the network operation and has a wide and centralized view of it.

Although Mininet provides its own SDN controller, we prefer to use **Opendaylight** as

we explain in Section 3.3, due to its preferred features as a control layer. This controller in particular is widely used in real world scenarios to study SDN networks.

One of the advantages provided by Opendaylight is an interface which allows the access to the controller's database. In addition, the Services Abstraction Layer (SAL) allows for the representation of network elements as objects, which simplifies the configuration and consulting of its data. For this reason, modules and APIs can be easily developed in order to allow for network services and provide an easy access to data related to the network.

Opendaylight controller is configured in a way that distinguishes between 2 types of information: Operational data and Configuration data. Operational data represents information on the current status of the network, such as link status or switch availability. Extracting this type of data is fundamentally the aim of the collection system. The Services Abstraction Layer (SAL) of the controller, mentioned previously, allows us to update this data in real time by using reports.

On the other hand, we have configuration data. We are not going to focus on this type of information. It is related to information which users push through REST API, for example to specify a host name or table routes, and we are not pushing any specific configuration for our purposes.

Apart from that, we have the Southbound interface of the controller which allows for the communication between the switches and the controller through Openflow protocol, mentioned in Section 2.2.3.

We are using Apache Karaf on Opendaylight to provide a friendly ecosystem for the controller, for example by providing a command line interface and web console. Karaf is an open source runtime environment which works on top of the controller. It allows us to install features, such as a web interface and any other deployment supports that simplify the management of the controller.

4.5 Ingestion layer

Selecting the data sources is one of the most important steps of the designing process. We use Logstash as the tool to gather data from the network. As we explained before in Section [3.4], we have two different data sources in our ingestion process: SDN controller and sflow collector. Making these periodic requests requires the use of Http poller logstash input plugging, mentioned previously in Section 2.4.1. We make these requests periodically to ensure the system works in a near real-time.

4.5.1 Opendaylight REST API

Opendaylight controller provides a REST API, which is used in this project to make queries via http requests and extract information. We can obtain access to all the data structures that have been defined in the controller. The aim is to get only data related to the network status and discard the rest. This data retrieved comes in a JSON format and wil be processed in the subsequent module.

In the first place we make this HTTP GET request:

http://10.0.1.12:8181/restconf/operational/network-topology:network-topology

The response to this query provides information related to the network topology. In other words, we extract how the devices are connected to each other and also IPs and MAC addresses from hosts and switches. In addition, we can find out the status and availability of these hosts on the network. One of the utilities of this data is, for example, to build a network topology diagram in our visualization module.

This a small piece of the information received in JSON format serves as an example:


```
"network-topology": {
    "topology": [
        {
            "topology-id": "flow:1",
            "node": [
                {
                    "node-id": "host:ea:5b:b7:07:4c:97",
                    "termination-point": [
                        {
                            "tp-id": "host:ea:5b:b7:07:4c:97"
                        }
                    ],
                    "host-tracker-service:addresses": [
                        {
                            "id": 0,
                            "mac": "ea:5b:b7:07:4c:97",
                            "last-seen": 1575042971306,
                            "ip": "10.0.0.1",
                            "first-seen": 1575042971306
                        }
                    ],
```


Figure 4.3: Data received from HTTP request.

Besides, we also collect data from the inventory manager making this other request:

http://10.0.1.12:8181/restconf/operational/opendaylight-inventory:nodes

This second request provides detailed information about the nodes on the network. We can find out how many interfaces per switch we have and how many packets and bytes are transmitted and received on every interface on every switch. In addition, we have data in reference to the availability of the links which connect the network elements. This information is very useful to create tables and graphs to include in our monitoring tool.

4.5.2 SFLOW Ingestion

The second data source in the ingestion process is the extraction of sflow data. Firstly, we have to indicate which switches on the network work as sflow agents. The sflow agents have the responsibility of sending sflow traffic information to the sflow collector. Therefore, we have five sflow agents because we have five switches in our network prototype. We have to execute the following command on the mininet container to configure the mentioned sflow agents on every switch.

```
sudo ovs-vsctl – –id=@sflow create sflow agent=eth0 target=10.0.1.16:6343 – set
bridge1 sflow=@sflow
```

Once we have defined the sflow agents and generate traffic within the network, we make use of the sflow-rt REST API to make requests and ingest sflow traffic in our system. However, before extracting, we have to push the format of the sflow datagrams, which means that we have to indicate first the information we want to receive from the application. By default it only sends information that we do not consider sufficient for our purposes. We modify it by executing the following command via CURL:

```
curl -H ”Content-Type:application/json” -X PUT –data
’”keys”:”ipsource,ipdestination,tcpsourceport,tcpdestinationport”,
”value”:”bytes”, ”log”:true’ http://localhost:8008/flow/flowgraph-
pair/json
```

Now, we can retrieve information at each periodic request and get data we need related to the active traffic on the network. There is information about IP source, IP destination, port source, port destination and bytes transmitted per flow. With the use of the periodic http request to the next URL, we are able to extract active flow information from the sflow-rt collector.

Once we have decided which sources we want to use in our model, we execute the following command to make Logstash start extracting information by using its Http poller plugin with the URLs mentioned previously:

```
bin/logstash -f pipeline/logstash.conf –path.data .
```

This command uses the logstash.conf file where we have all the mentioned configuration.

4.6 Processing layer

At this point, we have already decided and configured the data sources. We continue using Logstash for this process. The raw information comes in JSON format, and we have to select only the relevant fields and discard useless data. For example, since each node has many flow tables, some of them could be empty or have outdated entries. For this reason, we are exploiting the Logstash filter plugin capabilities to adapt the data for our monitoring requirements.

The information is represented in a nested JSON. This means that the information is disposed in "tree" format and there are array positions containing more arrays. The process of separating into different events these array fields is achieved by using the **split** logstash plugin.

```
split{
  field => "[network-topology][topology][0][link]"
}
mutate{
  add_field =>{
    "dest-node" => "%{[network-topology][topology][0][link][destination][dest-node]}"
    "dest-tp" => "%{[network-topology][topology][0][link][destination][dest-tp]}"
    "source-node" => "%{[network-topology][topology][0][link][source][source-node]}"
    "source-tp" => "%{[network-topology][topology][0][link][source][source-tp]}"
    "link-id" => "%{[network-topology][topology][0][link][link-id]}"
  }
}
```

Figure 4.4: Split and Mutate filter.

In the example represented in Figure 4.4, we separate the array named "link" into different events. Therefore, we will have one event per position in the "link" array referring to each link on the network topology.

Moveover, we use the **mutate** filter to add or modify fields. The path *[network-topology][topology][0][link][destination][dest-node]* refers to a field that contains an object with relevant information. Here is when we select the specific fields we want to add to our system among the huge amount of JSON data provided. We will use these fields later to create visualizations.

Almost every field is represented in *String* type by default. However, we can modify its object type by using the *convert* option, one of the options provided by the mutate filter. It is very commonly used for numerical values, such as the bytes received or the bytes transmitted field. This process is required to be able to represent numerical values in graphs.

Finally, we make use of the **grok** filter to extract unstructured fields represented in the same line. In other words, we have to use this filter when there are several fields contained in the same field space. We use it in this project to extract fields contained into the sflow datagrams.

4.7 Persistence layer

Now that we have two modules to ingest and process network information, we need the implementation of the Data Lake to add persistence functionality to the system. Therefore, the aim of the persistence layer is to store documents received from previous processes. In order to do it, we make use of Elasticsearch. As we explained in Section 2.4.2, Elasticsearch is an open-source application which stands out for its availability and flexibility on integration with the Elastic Stack.

Before receiving this data into Elasticsearch we have to specify first in Logstash the different indexes to classify the information. We have three indexes, one for each type of information: Nodes, Topology and Flows. We specify it on the logstash outputs.

We can access Elasticsearch on localhost port 9200, where we can look at the information received and consequently stored inside. For instance, we can look into detail the JSON data structured in "tree" format, to ensure how the fields are disposed for further processing.

Besides, it is possible to make specific queries to Elasticsearch by using the REST API provided. For example, we can search for information related to a particular index or delete the information stored.

4.8 Visualization module

This is the last conceptual layer which allows the end user to deepen the processed information and interact with it in different graphs. We use Kibana, mentioned in Section 2.4.3, for this process.

It provides a web interface where we can retrieve and show data stored in Elasticsearch. This interface offers end users the ability to make a query in any field by typing the interested value. The search results highlight the values found within all the documents in the time range selected.

One of the best characteristics of Kibana is that we can save searches. In other words, we can create our own search by selecting or discarding the fields we need from a list of available fields, which we have defined previously in the processing module. This is very interesting when we have to create visualizations because sometimes we only need specific fields and these saved searches make it easier.

4.8.1 Kibana Network Pluging

Additionally, we have installed a Kibana plugin used for the network topology representation in our monitoring tool. Unfortunately, the *official* visualization to represent networks in Kibana is only available in a paid version. This is why we have to look for other alternatives like this plugin [3].

This plugin has the same functionality and we can create a network topology visualization using the data provided by the controller.

CHAPTER 5

Results and Use Cases

Once we have developed the required modules of the proposed architecture, we run traffic and network simulations to collect and process data. After that, we represent all the information in the form of visualizations for a possible end user that needs to monitor an SDN network.

5.1 Introduction

In this chapter, we are going to evaluate the results obtained in our system. First, in Section 5.2, we are going to show two dashboards referring the network status Then, we present some use cases of our monitoring tool in Section 5.3.

5.2 Results

We have implemented a monitoring tool for SDN networks using Kibana. The results are shown in two dashboards. The first one, **General Dashboard**, includes a general view of the status of the network. In addition, we have the **Specific Dashboard**, in which we can be informed on traffic volumes on a specific switch and its interfaces.

5.2.1 General Dashboard

This is the main dashboard in which we represent vital information about the network. Firstly, we show a topology diagram made with the network plugin mentioned in Section 4.8.1 and shown in Figure 5.1.

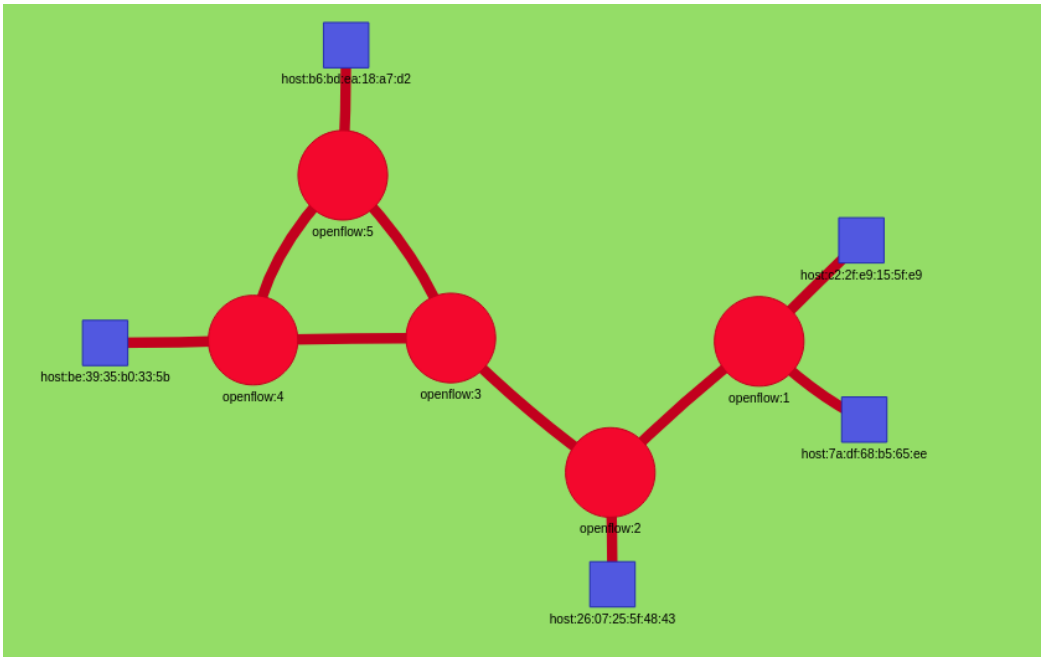

Figure 5.1: Topology.

As we can see, it matches with the proposed topology represented in Figure 4.2. We can zoom in and zoom out, so we can read the label of every device. This network representation changes if a modification on the network topology is detected, for example if we shut down one of the hosts. Below it, we have this table in which we can check the availability of the hosts represented and their IP and MAC address:

HOSTS INFO & STATUS

HOST_IP	HOST_MAC	ACTIVE
10.0.0.1	ae:33:ea:74:17:20	true
10.0.0.2	7e:4c:5d:c4:d5:7f	true
10.0.0.3	de:28:08:b3:0f:ba	true
10.0.0.4	56:c3:0b:00:9e:72	true
10.0.0.5	26:54:bf:ae:4c:ac	true

Figure 5.2: Hosts availability.

In addition, we have this other visualization by its side where the switches and the number of interfaces connected to each one are shown:

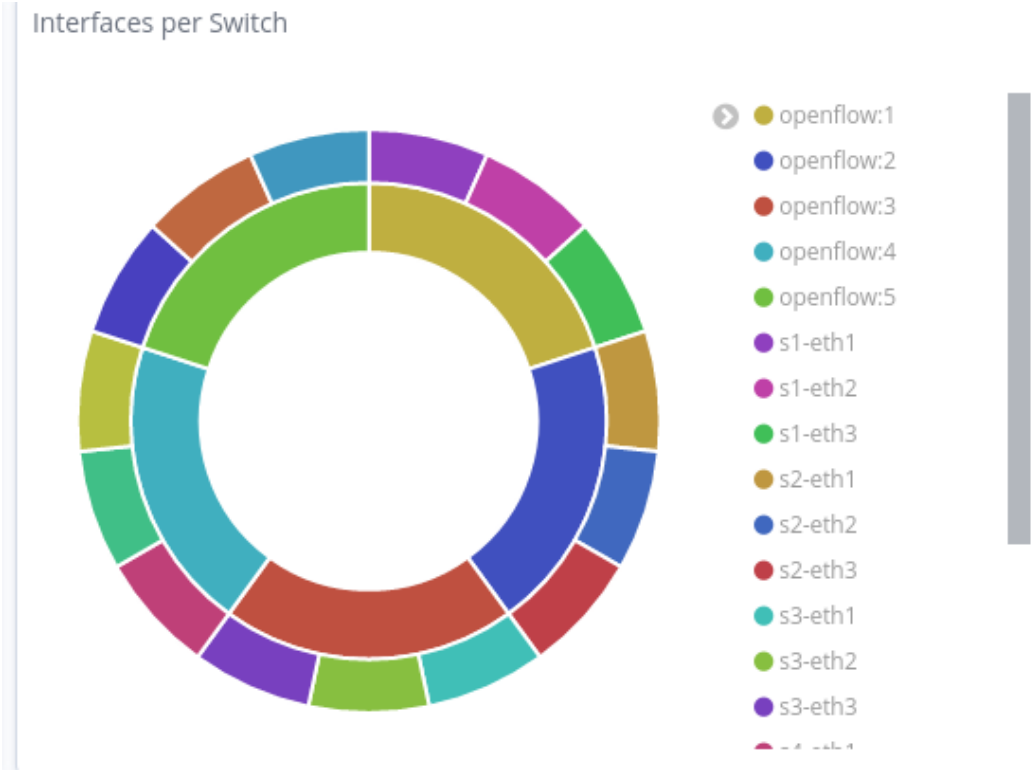

Figure 5.3: Interfaces per switch.

In this particular case, there is the same number of interfaces per switch and the pie chart looks symmetric. On the right, there is a legend information list which explains the visualization by itself.

This dashboard also includes information related to traffic flows. This information comes from the sflow collector, and we consider it vital in a monitoring tool.

In this particular example, we have generated traffic using Iperf, installed previously in the network container. Specifically, we have generated two TCP traffic flows. Therefore, there are two symmetric flows, because TCP sends ACKs for confirmation.

Regarding this information there are four visualizations:

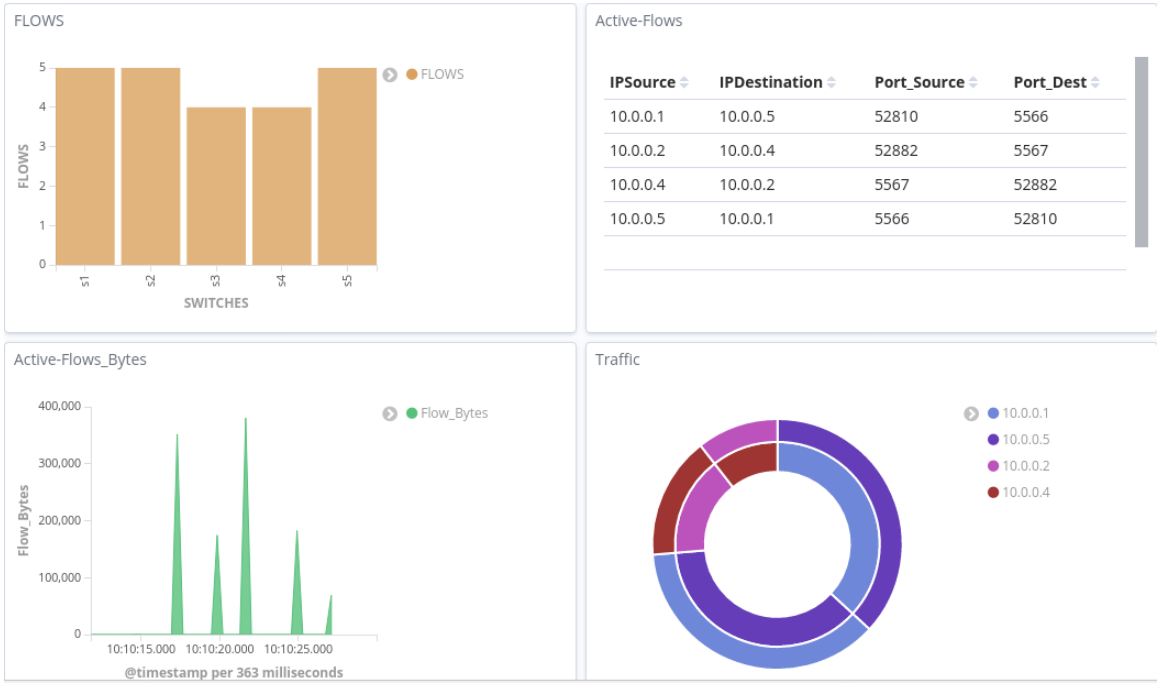

Figure 5.4: Flows information.

In the top left corner we can check the total flows per switch by looking at the bar chart. Just to its right, there are the active flows being transmitted within the network. As we explained before, there is one for each TCP traffic flow and their confirmation.

In the bottom left corner, we have represented the traffic flows over time. Is is possible to zoom in and zoom out changing the time window. To its right, there is a pie chart to show the active flows and their IPs in which we can check the amount of traffic for each flow. The inside swept circle represents the IP source, and the one from outside the IP destination. This information is shown by hovering the cursor over the area.

Finally, there are two more visualizations, illustrated in Figure 5.5. The first one, shows a representation revealing the relation between packets looked up and packets matched per switch. In this particular case, the relation is almost the same and nearly every packet matches in a switch. The proposed network is not connected to the Internet, and the traffic generated within it is locked inside. This is why almost every packet matches.

Next to it is the last representation in this dashboard. It shows a table containing data related to link and interface status. Therefore, we can check link availability in real time.

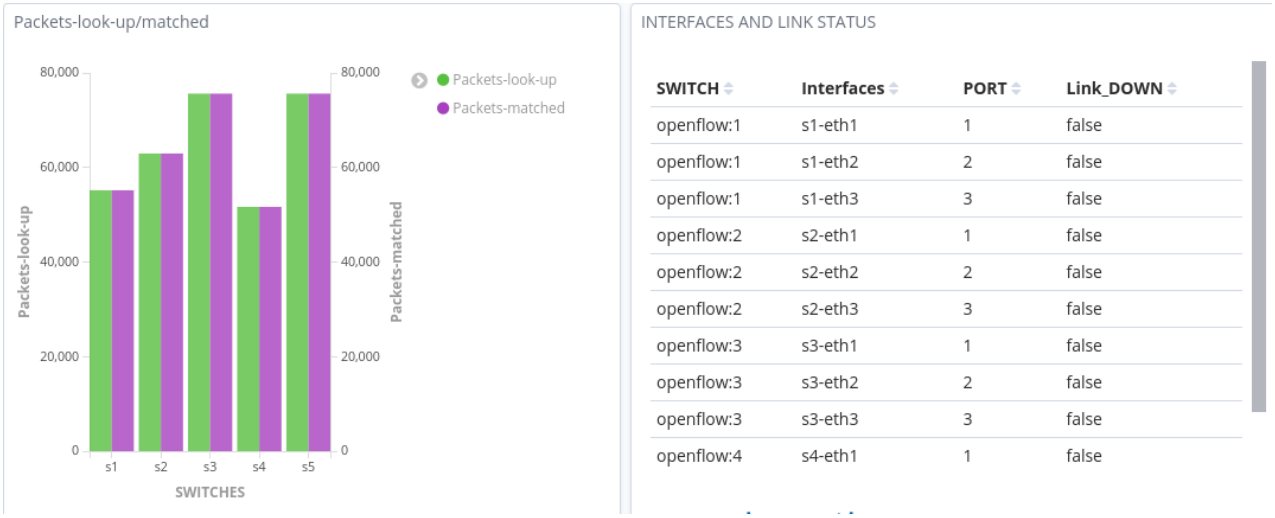

Figure 5.5: Packets look up/matched and link availability.

As a consequence, this dashboard gives the end user a general view of the network status. There, the essential information we could retrieve from the SDN controller and the sflow collector are shown in it, such as topology, traffic or host and link availability.

5.2.2 Specific Dashboard

This second dashboard represents information related to the volume of traffic. The idea of this dashboard is to first choose a specific switch. We can select one of them by clicking on the table or one of the bars in the bar chart. When we select a switch, a filter is applied to all the information, so now we only have information about that particular switch. However, the information from every switch will be shown if we do not select anything, as we can see in Figure 5.6.

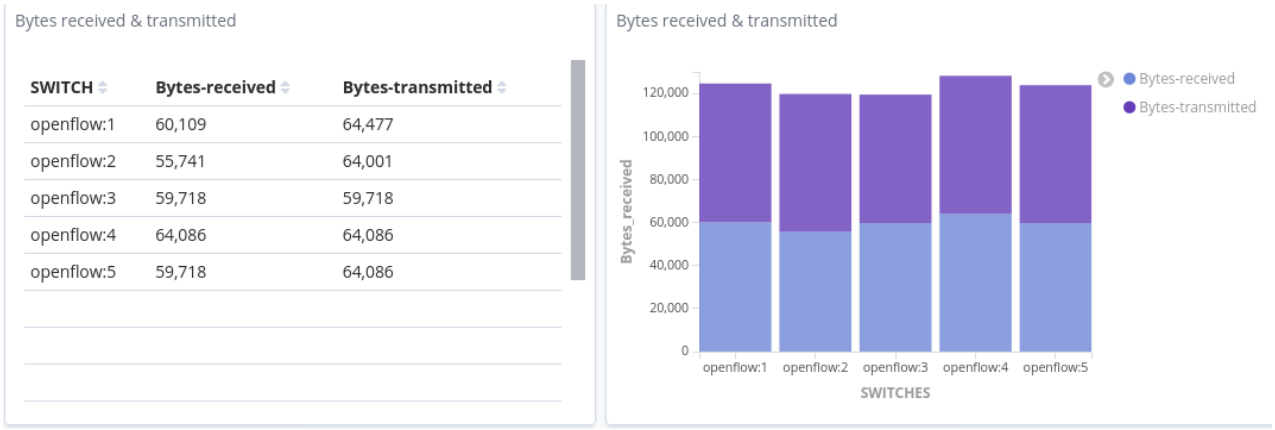

Figure 5.6: Total bytes received and transmitted.

Here we can check the number of bytes received and transmitted per switch. The chart on the right shows the same information represented in the table, but displayed in a bar chart. Therefore, we can see the relationship and compare both numbers easily.

The next visualization shows the traffic volume per interface. This is why we should select one switch at the beginning, so we can delve deeper in this specific switch and check its interfaces. Alternatively, is it possible to apply other filters so we can compare different switch interfaces.

This particular dashboard, and especially these kinds of visualizations can be used depending on the needs of the end user by applying filters to the information disposed.

In the next figure we have an example of switch Openflow:4 and its interfaces:

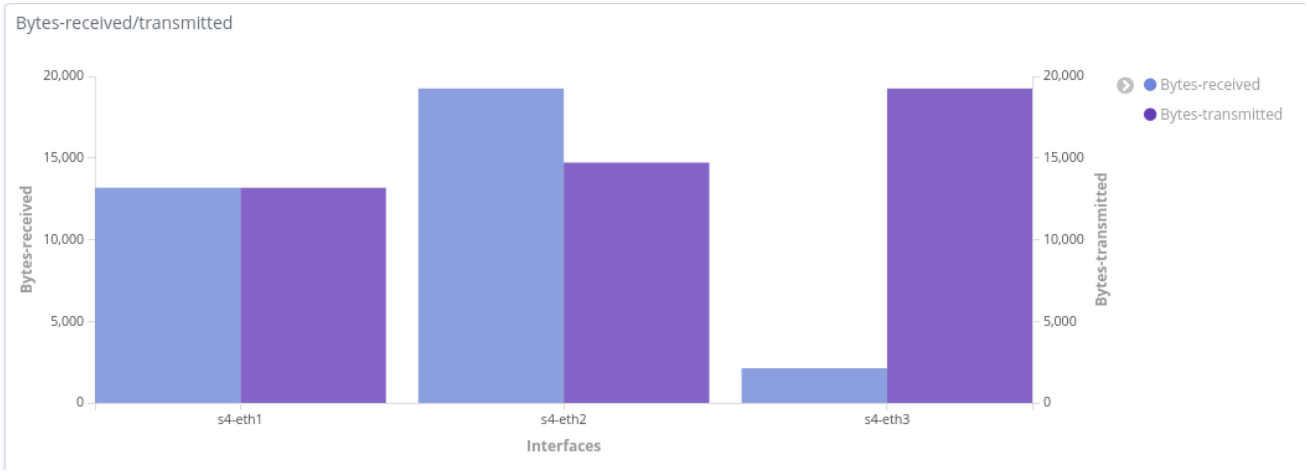

Figure 5.7: Bytes received and transmitted on Openflow:4 interfaces.

There is also an additional bar chart where we show the packets transmitted and received. It maintains the same relationship than bytes transmitted and received, since packets are composed of bytes.

In addition, we add a table showing this information. In Figure 5.8 we illustrate an example of the table in which we display the information numerically. We gather the information from the two visualizations so that we can compare them numerically.

Bytes and Packets

Interfaces	Bytes-received	Bytes-transmitted	Packets-received	Packets-transmitted
s1-eth1	2,196	29,900	30	368
s1-eth2	2,196	29,900	30	368
s1-eth3	28,290	26,596	345	320
s2-eth1	26,596	28,290	320	345
s2-eth2	2,196	29,816	30	366
s2-eth3	26,680	28,206	322	343
s3-eth1	28,206	26,680	343	322
s3-eth2	23,460	23,460	276	276
s3-eth3	26,680	28,206	322	343

Figure 5.8: Bytes received and transmitted per interface.

Finally, we make use of two more visualizations where we can look at the fluctuations in traffic transmitted and received. It is possible to zoom in or zoom out so end user can focus on a specific time window.

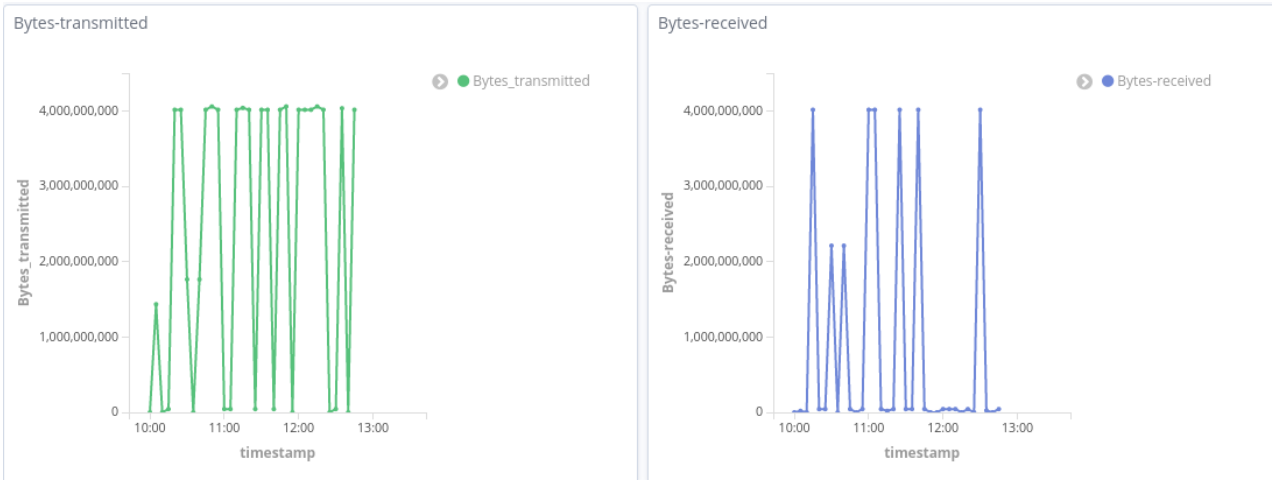

Figure 5.9: Traffic variations.

5.3 Use Cases Examples

In this section, we expose some uses and applications of the proposed system. The end user of this monitoring tool, such as network operators, can use the system for many purposes. We describe a few of them. However, there are multiple uses for this monitoring tool, depending on the uses the end user wants to give it. For instance, knowing the topology, traffic flows or host availability, to name a few.

5.3.1 Detection of link failures

The first test consists of simulating a link failure. The system must detect this failure and show it on the visualization module. Therefore, the end user can find out that link is not prepared to transmit or receive traffic.

In the first place, we have to indicate that one link must stop working on our emulated network . We choose the link that connects host H1 to switch S1. We have to execute the following command on mininet CLI to achieve it:

```
link h1 s1 down
```

We can observe the results in Figure 5.10. After executing this command, **h1** became unreachable.

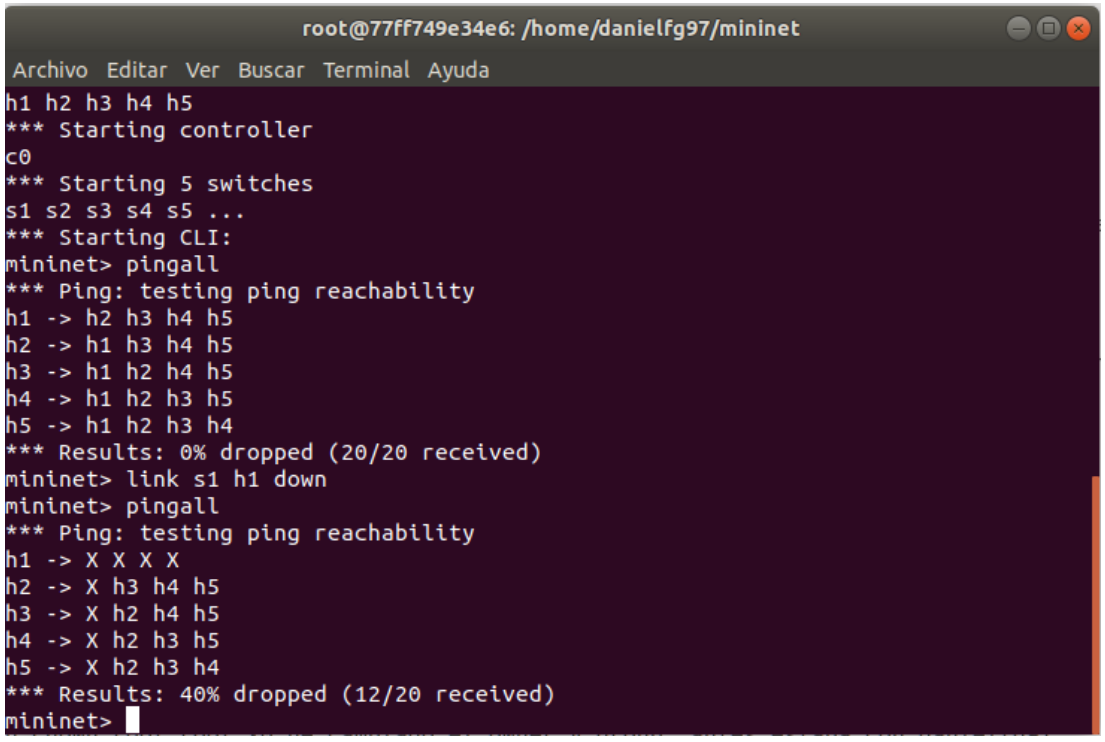

Figure 5.10: Link down.

Now, if we look at the visualization module, we can observe that interface **s1-eth1** is not available on the network.

As we can see in Figure 5.11, the system uploads the information represented in real time, so we can be aware of any possible modification in the network.

INTERFACES AND LINK STATUS

SWITCH	Interfaces	PORT	Link_DOWN
openflow:1	s1-eth1	1	true
openflow:1	s1-eth2	2	false
openflow:1	s1-eth3	3	false
openflow:2	s2-eth1	1	false
openflow:2	s2-eth2	2	false
openflow:2	s2-eth3	3	false
openflow:3	s3-eth1	1	false
openflow:3	s3-eth2	2	false
openflow:3	s3-eth3	3	false
openflow:4	s4-eth1	1	false

Figure 5.11: System detecting link failures.

5.3.2 Detection of overloaded interfaces

One of the most interesting applications is to study the possible overload on interfaces. It may be the case where one link is very overloaded and another barely used. In these cases we can look at the traffic graphs represented in the specific dashboard and compare the traffic transmitted or received in the switches interfaces. For instance, this might happen on more complex networks. By looking at the bar chart we can check if any interface is not being used. This information is useful to find out if we have some troubles when we generated our topology. For instance, we can look at Figure 5.7 and realize that interface 3 is not receiving almost any traffic. We should check this link in our topology and maybe make some changes when proceeding. This can be scaled to other situations and other network models.

In addition, with the numeric information provided in tables, we can find out the volume of traffic per switch, and see if any of them is working more than the others. Of course, we should combine this information with our topology design.

5.3.3 Acknowledgement of traffic flows

It is very important to be aware of the traffic on the network. With this monitoring tool we can detect how many flows are being transmitted on the network, and also which IPs are involved in them.

In the following test, we generate traffic simulating a possible situation in the network. Our fifth host, **h5**, plays the server role while **h1,h2** and **h3** are playing the client role. In other words, three symmetric flows have to be directed to our server h5 and the system must illustrate this information.

Regarding this information, in the first place we have to look at the General Dashboard. As we can see in Figure 5.12 there are 3 symmetric flows pointing to h5 (10.0.0.5). The red color on the pie chart represents h5 as the main target declared in this test.

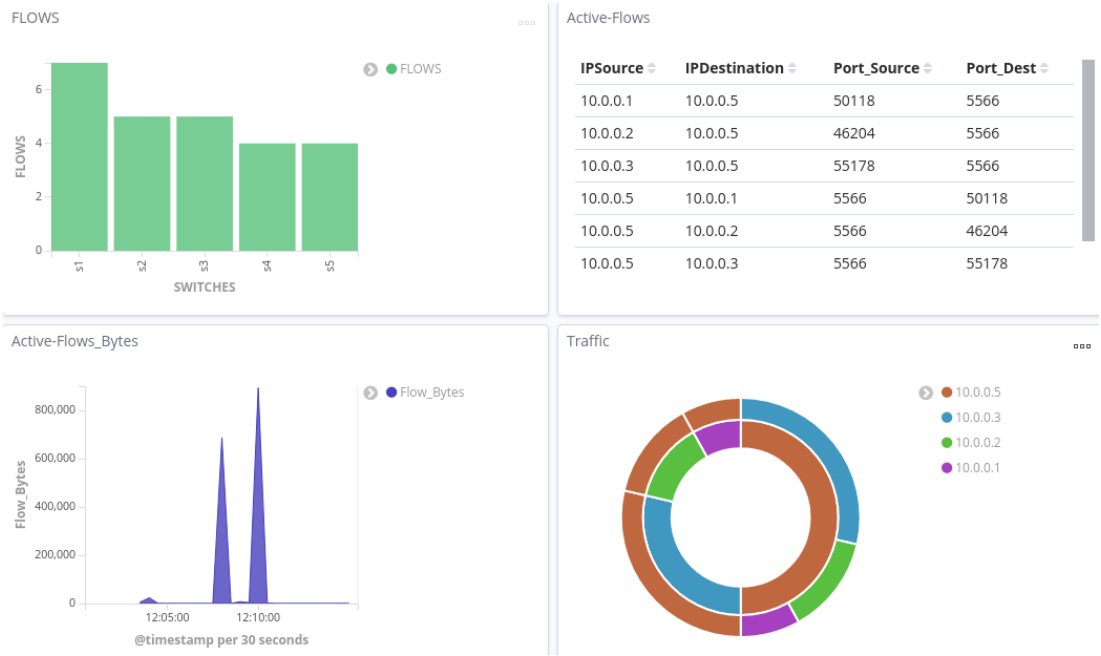

Figure 5.12: Traffic test.

Besides, since we have generated a considerable amount of data, we can look at the specific dashboard and check that around 12:13 P.M there was a traffic peak. Both graphs look almost symmetric, because this network model has no outside connection and the bytes transmitted on one side are received on the other.

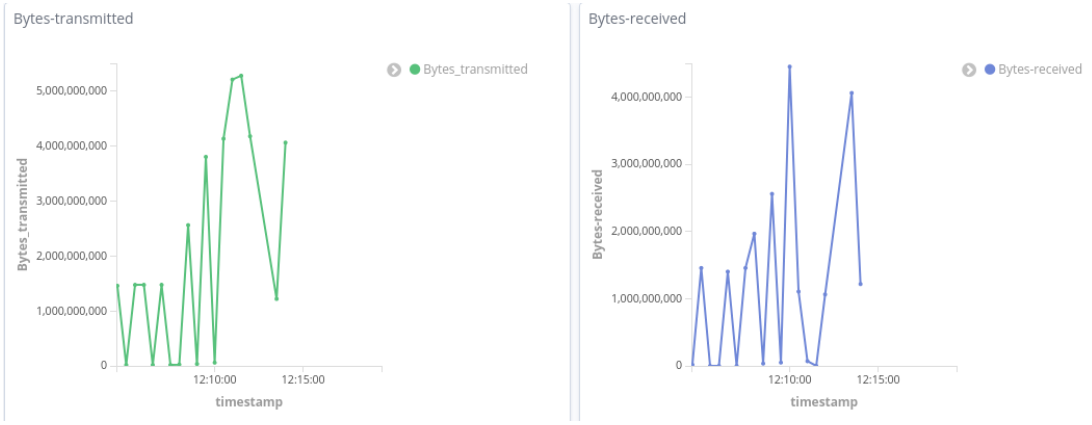

Figure 5.13: Traffic peak.

CHAPTER 6

Conclusions and Future Work

In this project, we have proposed and prototyped a solution for monitoring SDN networks. The implementation has been possible by making use of network emulation tools and big data technologies. In this chapter, we present the conclusions obtained from such work and propose possible research lines for the future.

6.1 Conclusions

New technologies such as SDN provide a number of benefits in the virtualisation and management of network services. For this reason, monitoring these kinds of networks is essential and it allows us to detect faults, watch out for performance and works as a way to guarantee quality of service.

Once we have emulated a SDN network with an external controller and added an sflow collector to our system, we have to select adequate data sources and process the information collected. Then, we start creating visualizations which will be grouped in dashboards. As a result, the end user has to be able to interact with them and be informed about the network operation and status in real time.

To perform this project, we have taken advantage of different technologies, mentioned and explained in Chapter 2. Then, we have designed an architecture and we have implemented a prototype using these technologies, such as Docker, which embed the whole project.

Finally, we have illustrated the results by representing the collected and processed information in interactive dashboards and we have done some tests to show the good functioning of the system. We demonstrated that the proposed monitoring tool works in real time for SDN networks, and therefore we fulfilled the purpose of the project.

6.2 Future Work

Although this project meets its goals, it is always possible to implement new functionalities and improve it. The main objective of a monitoring tool is to alert the end user about a possible failure. This could be one of the possible future improvements to the system, since the initial aim of the project does not cover this functionality.

When the monitoring platform detects, for example, a link failure, we would like to be informed about it. The system should send an email to the administrator containing this information, especially what happened and when, so the network administrator can be aware of the problem without looking at the dashboards. As we mentioned before, this implementation could be a whole other project and is not covered in the initial objective of this one, but it is a well-founded initiative we could implement in a future work.

On the other hand, this monitoring tool could be part of a much bigger system, for example in which it could be possible to self-manage telecommunication networks based on different artificial intelligence and self-decision systems. We could collect information that helps those systems make their decisions. For instance, if a residential area holds more connections, we have to keep in mind the switch that holds these connections is more susceptible to underwent a overload. As we said, the proposed system in this document could be part of a big project that combine several variables to help making decisions, and therefore, to self-manage SDN networks.

Appendix

APPENDIX A

Impact of this Project

Computer networks are present in almost every aspect of our everyday life and we depend on their good functioning. In fact, we could say that computer networks are big part of our life and simplify global communication. Almost every company needs an network infrastructure to provide their services. Therefore, we need a system always available, and try to avoid a lost of service. For this reason, the proposed system is useful for being aware of failures than could affect the network operation.

In this appendix, we explain the social, economic, environmental and ethical implications of such system.

A.1 Introduction

In this appendix, we consider the social, economic and environmental impact that this project could have in Sections A.2, A.3 and A.4, respectively. Besides, we deal with the possible ethical and professional implications of such project in Section A.5.

A.2 Social Impact

Faults on computer networks could have a serious impact in our everyday life. If we are relaxing and making use of a computer network, for example, consuming any type of content on the Internet like video-on-demand services, a computer network failure could affect our stress levels. This could also affect our work, if we lose connection to the internet, it is possible that the affected person could not be able to continue working.

We have to remark also the possibility of the creation of new jobs related to the deployment and maintaining of systems like the one developed in this project, and the creation of jobs related to the extension and developing of the mentioned system. As we explained in Section [6.2], there is still work to do to improve this project.

A.3 Economic Impact

Especially, the principal economic impact of this project affects the availability of service. Therefore, it could affect a possible lack of trust in the provider of such service and a probable lost of income at the company.

Thanks to monitoring tools we can check the status of the network and their devices connected. If we provide a good quality service, it is probable that more clients want to join and spend their money on internet services. In addition, if we detect failures there will be a technician hired to fix it and maintain the network, and therefore it improves economy creating jobs.

We can now develop applications that will work in any computer network and trust in the network infrastructure.

A.4 Environmental Impact

There are several situations that could affect computer networks. For this reason, monitoring networks is essential to detect this possible failures in time. However, the proposed system does not have big environmental impact. At least, we could say that it spend electricity, because it is run inside computers, and therefore consume energy that comes with a possible contamination of the environment. For instance, if we run this monitoring tool in thousands of computers we will spend a lot of energy, and if that energy is generated with petrol, it will be pollutant to the environment.

A.5 Ethical and Professional Implications

The ethical implications of this project are basically the ethical implications of collecting data from the network controller about every switch on the network, and knowing the IPs involved in traffic on it. This could affect the confidentiality of the information, since we can find out the IP from source and from destination within the network. All this information is stored in our database, and it is very easy to search for data related to a specific IP. In this field there is always a controversy with the collected data, and who can access to it. However, we use this information just to be informed and not to any unethical issues.

Finally, there is also a professional implication because there could be people hired just to maintain and fix network issues or improve the proposed system. As we mentioned before in the economic impact, this could create jobs. There is always some aspects that can be improved, and therefore people hired to work on it.

APPENDIX B

Cost of the System

There are some cost related to the design and development of this project, most of them associated with the salaries of the developers or hardware needs. Besides, we would also need money for licences in case we want to commercialize our software. We would also have to be aware of taxes, since it is something we put available in the market for selling.

B.1 Introduction

The objective of this appendix is to evaluate the possible expenses caused by the development of this system Firstly, we depict the costs in the hardware needed for our system to run in Section B.2. Then, we estimate the costs related to the personnel needed to implement and maintain this project in Section B.3. Next, in Section B.4 we specify the funds needed for software licences. Finally, we have in consideration the probable taxation involved in case we want to sell this software in Section B.5.

B.2 Physical Resources

In order to run our system, we need a powerful computing machine, sufficient enough to run every module contained in a docker container. To develop this system we made use of a computer with the following characteristics:

- **CPU**: Intel i7 processor
- **Hard Disk**: 500 GB
- **RAM**: 8 GB
- **SO**: Ubuntu 18.04

After developing the project in this machine, our experience is that maybe we would have needed more RAM memory. We need memory to run with good quality several docker containers at the same time, 8 GB is sufficient but we would recommend at least 16 GB.

The average cost of a machine with this features is around 1000€ in the market. We need good computing resources to run the whole system. The cost will rise, but also the quality of the service this project provide.

B.3 Human Resources

In this section, we estimate how much time we spend in the designing and development process, and therefore how much would it cost to implement it. We make a prediction using the average salary of a Software developer to calculate costs.

We estimate around 400 hours of work spent on the project, estimating 28 weeks, 4 times a week, and 3-4 hours per day. We should keep in mind the hours when searching for information, designing the prototype, writing code, test the result and solve errors. Besides, we had to print that information in a document and we spent around a month on it. However, we do not include that amount of time in costs.

The average salary of a junior developer is around 1500 euros/month. Therefore, keeping in mind the hours mentioned before, developing this prototype would cost around 5500€. In addition, there has to be personal to maintain the project and this would add around 20000€/year if we hire a full-time engineer for this purpose. However, these are wide approximations, and the cost would vary in reality.

B.4 Licences

Most of the software used in the developing of this project is open-source software, such as Docker or the Elastic Stack and therefore there are no cost in licences. However, we could pay Kibana and Elasticsearch paid versions, and this implies 712 dollars per year. This extended version provide new visualizations, such as the one to create a topology diagram, but we made use of an external Kibana plugin to achieve similar results.

B.5 Taxes

In the case we want to sell the software developed in this project, we would have to add taxes costs. According to [4], there is a tax of 15% over the final price regarding the Spanish law. Therefore, using the previous estimations, the final prize would be 6325€ with taxes.

Bibliography

[1] Number of devices connected. https://technology.informa.com/596542/number-of-connected-iot-devices-will-surge-to-125-billion-by-2030-ihs-markit-say Accessed: 2019-05-11.

[2] Osi levels. https://www.webopedia.com/quick_ref/OSI_Layers.asp. Accessed: 2019-05-11.

[3] Kibana network plugin. https://github.com/dlumbrer/kbn_network. Accessed: 2020-01-02.

[4] Francisco de la Torre Díaz. La tributación del software en el IRNR. Algunos aspectos conflictivos. *Cuadernos de Formación*, 10/2010:239–249, 2010.

[5] Yongning Tang, Guang Cheng, Zhiwei Xu, Feng Chen, Khalid Elmansor, and Yangxuan Wu. Automatic belief network modeling via policy inference for sdn fault localization. *Journal of Internet Services and Applications*, 7(1):1, 2016.

[6] PwC. The software-defined carrier: How extending network virtualisation architecture into IT BSS/OSS architectures opens up transformational opportunities for telecom and cable operators. https://www.pwc.com/gx/en/industries/communications/publications/communications-review/assets/communications-review-the-software-defined-carrier.pdf, 2016. Accesed: 2019-11-27.

[7] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6. IEEE, 2014.

[8] Dilpreet Singh and Chandan K Reddy. A survey on platforms for big data analytics. *Journal of Big Data*, 2(1):8, 2015.

[9] Naga Katta, Haoyu Zhang, Michael Freedman, and Jennifer Rexford. Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 4. ACM, 2015.

[10] Mininet. http://mininet.org/. Accessed: 2019-11-09.

[11] Elasticsearch. https://www.elastic.co/products/elasticsearch. Accessed: 2019-11-09.

[12] Hadoop. http://hadoop.apache.org/. Accessed: 2019-11-09.

[13] The OpenDaylight Project, Inc. Opendaylight. https://www.opendaylight.org/. Accessed: 2019-11-09.

[14] José Sánchez, Imen Grida Ben Yahia, and Noël Crespi. Poster: Self-healing mechanisms for software-defined networks. *arXiv preprint arXiv:1507.02952*, 2015.

[15] Cisco VNI. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021. *Cisco White Papers*, 2016.

[16] Cisco. The zettabyte era: Trends and analysis. *Cisco White Papers*, 2016.

[17] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. Mininet as software defined networking testing platform. In *International Conference on Communication, Computing & Systems (ICCCS)*, pages 139–42, 2014.

[18] Markets and Markets. Software-Defined Networking and Network Function Virtualization Market by Component (Solution (Software (Controller, and Application Software), Physical Appliances), and Service), End-User, and Region - Global forecast to 2022, 2017.

[19] Accenture. Network transformation survey 2015, final results. https://www.accenture.com/t20170416T224033Z__w__/us-en/_acnmedia/Accenture/Conversion-Assets/DotCom/Documents/Global/PDF/Indurties_17/Accenture-Network-Transformation-Survey-2015.pdf, 2015. Accessed: 2019-11-27.

[20] Accenture CMT Digital Consumer Survey 2015. Network transformation survey 2015, final results. Technical report, Accenture, 2015.

[21] Stewart Baines. Enterprises want sdn to build flexible networks. https://www.orange-business.com/en/blogs/connecting-technology/networks/enterprises-want-sdn-to-build-flexible-networks, 2015. Accessed: 2019-11-27.

[22] Stewart Baines. Enterprises want SDN to build flexible networks. Technical report, Orange, June 2015.

[23] Sean Vig. Network congestion as an emergent phenomena in internet traffic. 2011.

[24] Steve Alexander. When networks hit the wall. https://www.networkworld.com/article/3221333/lan-wan/when-networks-hit-the-wall.html, 2017. Accesed: 2019-11-28.

[25] Cisco. Cisco Global Cloud Index: Forecast and Methodology, 2015–2020. *Cisco White Papers*, 2015.

[26] Allan Vidal. Flexible networking hot trends at SIGCOMM 2016. *Ericsson Research Blog*, 2016.

[27] Nokia. 7950 xrs-xc: Paving the path to petabit routing. https://networks.nokia.com/solutions/7950-xrs-xc-core-router, 2017. Accessed: 2018-07-03.

[28] Orange Business Services. Enterprises want sdn to build flexible networks. https://www.orange-business.com/en/blogs/connecting-technology/networks/enterprises-want-sdn-to-build-flexible-networks, 2015. Accessed: 2019-09-03.

[29] Cisco. A successful digital business needs an agile network. https://www.cisco.com/c/dam/en/us/solutions/collateral/data-center-virtualization/unified-fabric/agile-network.pdf. Accessed: 2019-09-03.

[30] Deloitte. Disrupting telco business through sdn / nfv. https://www2.deloitte.com/es/es/pages/technology-media-and-telecommunications/articles/disrupting-telco-business-through-SDN-NFV.html. Accessed: 2019-09-03.

[31] Shamshad Lakho, Akhtar Hussain Jalbani, Muhammad Saleem Vighio, Imran Ali Memon, Saima Siraj Soomro, et al. Decision support system for hepatitis disease diagnosis using bayesian network. *Sukkur IBA Journal of Computing and Mathematical Sciences*, 1(2):11–19, 2017.

[32] Sun Jin, Changhui Liu, Xinmin Lai, Fei Li, and Bo He. Bayesian network approach for ceramic shell deformation fault diagnosis in the investment casting process. *The International Journal of Advanced Manufacturing Technology*, 88(1-4):663–674, 2017.

[33] Salma Ktari, Stefano Secci, and Damien Lavaux. Bayesian diagnosis and reliability analysis of private mobile radio networks. In *Computers and Communications (ISCC), 2017 IEEE Symposium on*, pages 1245–1250. IEEE, 2017.

[34] Fernando Perez, Brian E Granger, and John D Hunter. Python: an ecosystem for scientific computing. *Computing in Science & Engineering*, 13(2):13–21, 2011.

[35] Pox controller. https://github.com/noxrepo/pox. Accessed: 2019-13-09.

[36] Socat. http://www.dest-unreach.org/socat/doc/socat.html. Accessed: 2019-11-09.

[37] Wine. https://wiki.winehq.org/Main_Page. Accessed: 2020-14-03.

[38] Karaf. https://karaf.apache.org/. Accessed: 2019-14-09.

[39] Pgmpy. http://pgmpy.org/. Accessed: 2019-23-04.

[40] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[41] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open vswitch. In *NSDI*, pages 117–130, 2015.

[42] VR Benjamins et al. Problem-solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research and Applications*, 8, 1996.

[43] How can i use bibtex to cite a web page? - tex - latex stack exchange. https://tex.stackexchange.com/questions/3587/how-can-i-use-bibtex-to-cite-a-web-page. (Accessed on 03/03/2020).

[44] Logstash. https://www.elastic.co/products/logstash. Accessed: 2019-05-09.

[45] Bruno Lecoutre and Jacques Poitevineau. The significance test controversy revisited. In *The Significance Test Controversy Revisited*, pages 49–62. Springer, 2014.

[46] Logstash filter. `https://discuss.elastic.co/t/add-field-from-json-logstash-filter/69364`. Accessed: 2020-01-02.