universidade de aveiro deti departamento de eletrónica, telecomunicações e informática

SOFTWARE DEFINED NETWORKS MONITORING SYSTEM

PROJECT IN COMPUTER AND INFORMATICS ENGINEERING

AUTHORS

93444 David José Araújo Ferreira

103574 Guilherme João Dos Santos Craveiro

89119 João Tomás Pires Machado

SUPERVISORS

Professor Daniel Corujo

Professor José Quevedo

David Santos

Rui Miguel Silva

JUNE 2023

ABSTRACT

The ongoing merge between technology and all aspects of our lives seems to be ever so quickening, which could not be more evident as in one of its more prolific impacts: communication.

The rate, size, and flexibility at which we communicate have never been as large as it is today. Two of these requirements, namely size and rate (which translates into speed for devices), as of today, could be met with improvements to network devices' capabilities by making them capable of handling larger bandwidth and processing data faster. However, flexibility has been a largely overlooked issue when it comes to the evolution of the networks that are present in the systems.

For the question of adapting to a more flexible model, having the networks grown to such large sizes, the shift will be not only of device evolution but of concept. With the appearance of Software Defined Networks (SDN), the way we used to look at the network has changed. Old constants, such as the network growth signifying that the hardware (e.g. switches and routers) would have to be replaced with higher capacity models with more physical ports, and their configuration done with commands in the command line interface (CLI). SDN brings out new prospects to networks, empowering them, like many other aspects in the evolution, replacing many of these mechanisms (and even hardware itself) by software.

This report describes the research and experimentation made with the virtualization of an entire network, from hosts to network devices (switches and routers), and how that allows us to observe the network as a whole, allowing for network management automation and reactive/dynamic topologies in a way that was previously not possible.

A proof of concept is provided, evidencing the implementation of a monitoring system that, by taking advantage of all the actors being virtualized, is capable of interacting with them automatically to collect traffic flow data and reconfigure them in real time in reaction to the data collected.

ACKNOWLEDGMENTS

The team would like to express its sincere gratitude for the invaluable support and guidance provided by each of the professors throughout our group project. We are genuinely appreciative of your commitment, availability, and willingness to assist, which have been instrumental in the successful completion of this endeavor.

Your accessibility and timely responses to our inquiries, along with your expertise and valuable suggestions, have greatly enhanced our understanding of the subject matter. We are particularly grateful for your openness in allowing us to pursue the objectives that sparked our interest. Your trust in our judgment and independence has empowered us to explore and delve deeper into areas that captivated us.

The team acknowledges and values your extensive knowledge, expertise, and constructive feedback, which have contributed to refining the quality of our work. Your mentorship has not only improved our technical skills but has also fostered personal growth, instilling in each team member a sense of confidence and self-assurance.

We sincerely thank you for your unwavering support, guidance, and the time you dedicated to our project. Your mentorship has played a significant role in shaping our academic journey, and we are confident that the impact will resonate in our future endeavors.

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

5G	5th Generation Mobile Network
API	Application programming interface
ASIC	Application-specific integrated circuit
BMV2	Behavioral Model Version 2
CLI	Command line interface
DDoS	Distributed denial-of-service
DHCP	Dynamic Host Configuration Protocol
ECMP	Equal-cost multi-path routing
FPGA	Field-programmable gate array
GUI	Graphical User Interface
gRPC	Google Remote Procedure Call
IANA	Internet Assigned Numbers Authority
INT	In-band Network Telemetry
IP	Internet Protocol
JSON	JavaScript Object Notation
L2	Layer two
LAN	Local Access Network
LPM	Longest Prefix Match
MAC	Media Access Control
NIC	Network Interface Card
P4	Programming Protocol-independent Packet Processors
REST	Representational State Transfer
SDN	Software Defined Networks
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VLAN	Virtual LAN
VXLAN	Virtual Extensible LAN

CHAPTER 1

INTRODUCTION

With the advent of the fifth generation of mobile networks (5G), the focus has shifted from the presentation of faster and more capable devices, into what software could unlock, and how it could expand the use of existing devices, making them more flexible. With the growing trend of cloud and virtualization of services, how can this be applied to networks?

Traditionally, networking has been one of the most stable fields in technology, with its most significant shifts being the growth of communications bandwidth and speed. This succession of improvements from previous to current technology, although meaningful, does not present itself as a change in the stabilized concepts.

But this growth can present some difficulties, as anything large tends to lack flexibility and adaptability, and networks are not the exception. We have seen time and time again how the reconfiguration of a few devices can bring an entire network, or a large part of it, “to its knees” and how long and costly can be to bring it back to a fully operational state. We still rely on methods and protocols that served us, some of them for almost half a century, but were never designed to cope with the demands of today's environment.

Recently, with the introduction of SDNs, the concept of being tied up to hardware devices is overcome, adding as well the use of virtualized services that can be easily scalable and rapidly deployed to account for spikes in traffic or fast and steady growth. Although this is a considerable improvement over the existing infrastructure, the issue of configuration and reconfiguration still is not solved, as SDNs' focus on the creation of new virtual devices, but once these are deployed they will need to be configured and re-configured every time the topologies need to change(i.e., unless they are “cloned” from other devices already containing the intended configuration).

SDNs are a key aspect for deploying and operating integrally virtual networks, such as the ones that are dynamically created inside datacenters. To illustrate the concept, one can imagine a physical computer running a type 1 hypervisor (e.g. Proxmox, ESXi, etc) where there are multiple instances of virtual machines with different operating systems and different purposes. One could be running an Apache2 server while another one could be running a MySQL server, and if these were to communicate with each other inside the virtual substrate,

then that would be considered an SDN, and like traditional networks, there would be a need for network devices, like switches and routers, to mediate the communication between hosts. Besides physical, in a SDN, these devices could also be virtual, and in fact datacenters have the capability of hosting a vastly greater amount of virtual networking devices than physical ones. Since all of the networking devices are virtual, effectively just software, one could reprogram them through the usage of a language called P4.

P4 [1]is a domain-specific language for network devices, specifying how data plane devices (switches, Network Interface Cards (NICs), routers, filters, etc.) process packets. By doing this with a high-level programming language, once compiled, the resulting JavaScript Object Notation (JSON) configuration file can be pushed to every desired device resulting in the immediate reconfiguration of the way the device processes incoming packages. This has multiple use cases, from an L2 switch to a Firewall, and with the added advantage that, if the system has limited resources, one device can easily be converted to any other device according to the demands of the traffic flow at the time.

Another advantage of SDN is the utilization of OpenFlow[2]. Because network devices are just software, there is no need for each one to be configured and/or controlled separately. Instead, one could devise a centralized and common controller that has the power to configure each and all devices at any given time. To accomplish that, it needs to communicate with them and it does that with OpenFlow. OpenFlow is a protocol that allows a server to tell network switches where to send packets. In a conventional network, each switch has proprietary software that needs to be pre-configured, and tells the switch what to do. With OpenFlow, the packet-moving decisions can be centralized, so that the network can be programmed independently of the individual switches and data center gear. In a conventional switch, packet forwarding (the data path) and high-level routing (the control path) occur on the same device. An OpenFlow switch separates the data path from the control path. The data path portion resides on the switch itself; a separate controller makes high-level routing decisions. The switch and controller communicate by means of the OpenFlow protocol.

[1] P4 Language - https://p4.org/
[2] OpenFlow - https://opennetworking.org/

1.1 CONTEXT

This project was developed within the scope of researching the capabilities of integration between SDNs, its communication protocol, OpenFlow, and P4 programming language, in order to create a monitoring system that could be deployed to any new or existing SDN network operating with P4 in its devices.

It was developed at the Electronic, Telecommunication, and Informatics Department of the University of Aveiro, for the subject of Project In Computer and Informatics Engineering.

1.2 MOTIVATION

This project has the intent of applying the advantages of the flexibility from OpenFlow and P4 to a network architecture with the objectives of:

- **Better insight over the network health** – Since P4 is protocol independent, the possibility is given to the system administrator to specify proprietary package headers in order to gather data from every single frame in transit in the network and by doing so, observe how these frames are flowing through the network. Some interesting knowledge is hopping time, package size, bandwidth usage, network usage over time, or common source and destination addresses.
- **More dynamic topologies** – By having access to important key information about the type of traffic in the network, combined with the capability of being able to reconfigure network devices remotely in a programmatic way, not only the sysadmin can reconfigure the devices, but configuration could be automated by specifying rules that consider the result given by the processing of the collected data.

1.3 OBJECTIVES

The objective of this project is to create a user-friendly network monitoring/management dashboard. This dashboard should display graphically the metrics calculated from the data collected from the network.

From a dashboard, the user will be able to create and manage new/existing network devices, redefining their specification (e.g number of ports, Internet Protocol (IP) address, Media Access Control (MAC) address, which traffic to block). With this, the user can specify rules for the automated deployment of new devices. The user can specify new headers the network devices should recognize in order to collect new data. These new metrics can also be specified and how they are calculated. The dashboard should also be able to store historic traffic data from the traffic in the network, and with this be capable of predicting patterns in traffic flow and allocating, in advance, enough resources to cope with the expected type of traffic.

The goals can be subdivided into

- **Dashboard creation** - A web Graphical User Interface (GUI) from where the user will interact with the system and control a network.
- **P4 capable devices** - Creation of a device that can easily be deployed by supplying a P4 configuration file.
- **Controller** - Network controller that can connect to every device on the network, where it will collect the data sent from these nodes.
- **Header and metrics specification** - define a set of headers for collecting relevant data and which metrics to, and how to, calculate.

1.4 REPORT STRUCTURE

In addition to the introduction, this document has five more chapters. Chapter 2 describes the state of the art of similar research and the possible implementation of other monitoring systems using P4. Chapter 3 presents the elicitation requirements process and the system's requirements and architecture. In Chapter 4, the implementation of the working prototype is described in its three main parts: dashboard creation, P4Runtime REST API controller and infrastructure. At last, Chapter 5 makes an overview of the work done on this thesis, its main results, conclusions, and future work.

1.4.1 IMPORTANT REMARKS

Take note that in this report common expressions like "network device", "router" or "switch", mostly refer to their virtual counterparts since in this report we assume as a network, a

software-defined one, having said this, when referring to physical devices, this will be explicitly done.

CHAPTER 2

STATE OF THE ART

2.1 RELATED WORK

In this section we present all of the work studied in order to develop this project solution. All of the work presented here is of theoretical applications being that most of it is academic-related work. Most of this work also focuses on P4 programming, in particular in In-band Network Telemetry (INT). P4 language allows the user to create its own "type" of the network device by also allowing the design and creation of custom headers and "packets". This direct interaction with the pipeline by not only observing but altering the traffic itself opens up new possibilities when measuring telemetry data directly in each node of the network in comparison to other traditional options that are limited to the capture of traffic in specific points of the network.

These documents are not presented in any particular order since most of them are variations on the same general subject.

2.1.1 Improving Network Monitoring and Management with Programmable Data Planes [1]

The first work presented is not an academic work, not even a formal one, it is a post by the P4 organization on the Open Network Foundation website, where they introduce a broader view of the idea of managing and monitoring networks of virtual devices. It takes into context the growing virtualization of services and the advantages of the decoupling between the physical and virtual topologies but without losing the total connection between the two fields, thus providing one continuous environment.

It begins by mentioning some important network state information, that there are already methods for collection, and highlighting their shortcomings. It informs us that although this method provides valuable information, these methods still base themselves on client/server models where one calls the other periodically to inform of its state, which in situations of rapid network growth usually proves to be too much for CPU-based control planes, while the INT model on the other hand exports data plane information directly from it. It also highlighted how

the INT model provides us with access to information that previous models could not, like queue depths, packet drops, and routing/ECMP path selection.

Mentions are made of the unique properties of P4 that make it ideal to implement INT such as the capability to express multiple packet formats and header fields, provide primitives to perform meaningful analysis, and the ability to manipulate headers. An example of an implementation is provided in Figure 1.1, alongside more detailed information about its inner workings.

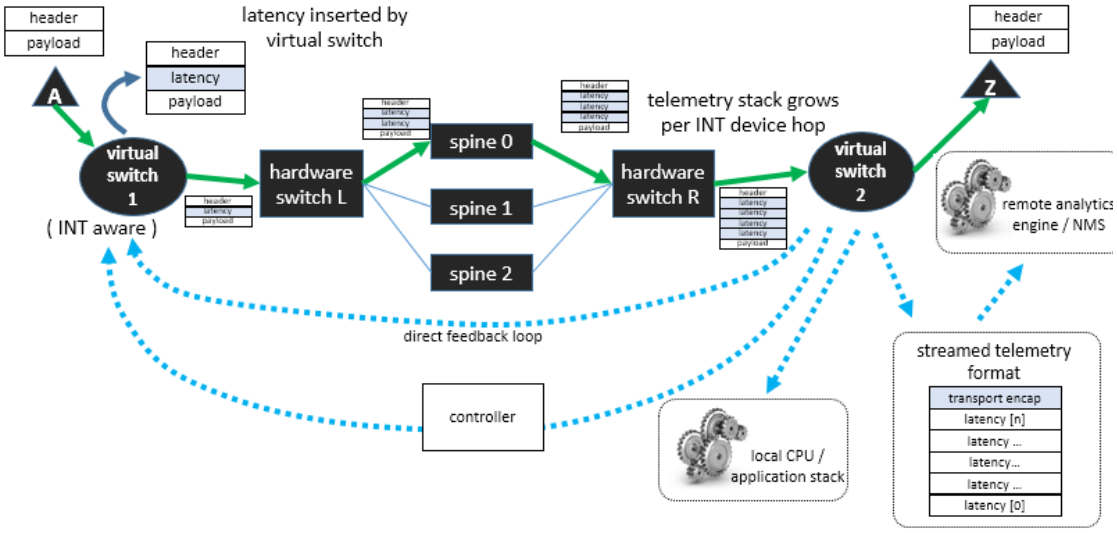

Figure 1.1 - Measuring and reporting end-to-end latency between virtual switches

source: [1]

2.1.2 Inband Network Telemetry: History, Impact and Future Direction [2]

After having read about INT and its close link with P4, this presentation offers a quick and insightful introduction to the technology. It exposed the key development principles as its performance being on par with typical software development, the fact that applications and endpoints can remain completely agnostic to INT, its flexibility in implementation, being capable of working over TCP, UDP, VxLAN/Geneve or IPv4 GRE. It also mentions some of the most important impacts, namely in simplified troubleshooting, Intelligent Path Selection as well as Congestion Control.

It also demonstrated conceptually how INT integrates with the data plane (see Figure 1.2) by how it collects, transports, and produces telemetry results in a network environment

between devices. It was a key document to illustrate how INT integrates with our desired solution and how we could utilize it.

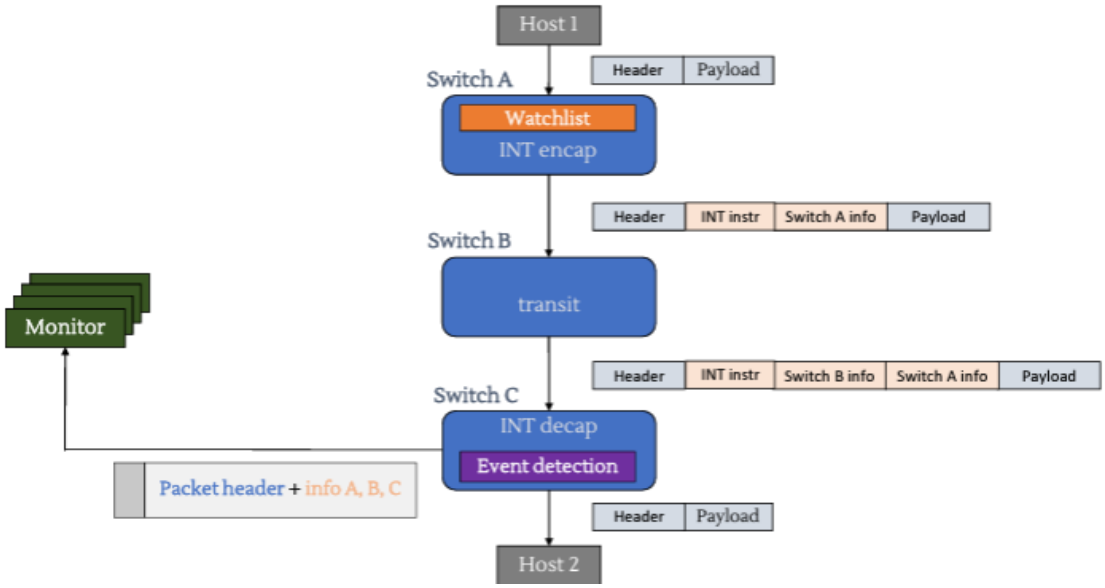

Figure 1.2 - Initial concepts of INT integration in frames and its role in telemetry collection and transport

source: [2]

2.1.3 In-band Network Telemetry Dataplane Specification, version 2.1 [3]

Probably the most technical of all of the reports, this report served more of a consultation guide in order to correctly implement INT given its specificities.

2.1.4 Implementation of Network Telemetry System With P4 Programming Language [4]

By far the most complete work consulted, and the subject being largely similar to our project, it served strongly as a base for the development of our work. In its first section, it goes in-depth to provide context by explaining what SDNs, OpenFlow, and P4 programming languages are, also taking a dive into the same major component of P4 development, like Behavioral Model and P4 Runtime API.

It also touches on the subject of INT, this time how it can be applied to P4 in a more concrete manner, what kind of modes can be used, and what information can be monitored with it.

The most relevant part of this work would be the last section, *Design and Implementation*, where the author goes in-depth, with demonstrations, into the implementation in a virtual network using Mininet[3] of network device programming via P4 and encapsulation of INT headers in the traffic flow in order to collect telemetry data.

2.1.5 Design and Development of Network Monitoring Strategies in P4-enabled Programmable Switches [5]

The focus of this report was on defining what type of difficulties traditional networks and how SDN can help solve these issues. We found certain particular aspects of this report quite interesting in the way that they helped us define some end goals along the development process.

Firstly, the authors propose three contributions to what they have identified as major issues, as follows: new algorithms to approximate some arithmetic operations in the programmable switches that are not supported by default in P4; development of five different monitoring tasks (partially) executable by programmable data planes: (i.) heavy-hitter detection to detect heavy flows with large packet counts; (ii.) flow cardinality estimation to estimate the number of distinct flows in the network; (iii.) network traffic entropy estimation to track the flow distribution; (iv.) total traffic volume estimation to know how many packets are flowing in the network; and (v.) volumetric DDoS attack detection to detect potential volumetric DDoS attacks by tracking entropy or flow cardinality sudden variations. The interaction between these is illustrated in Figure 1.3.

Although this report does not give any particular insight on how to implement said telemetry and data processing, it is quite useful as a concrete set of metrics to be applied in our system.

[3] Mininet - www.mininet.org

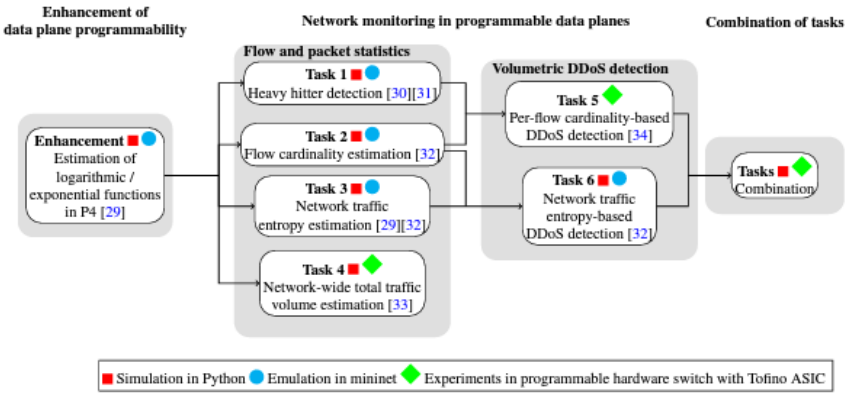

Figure 1.3 - Scheme of the report

source: [5]

2.2 TECHNOLOGY

In this section we take a look at the technologies used in this project, we will describe their main features and why they were chosen.

P4

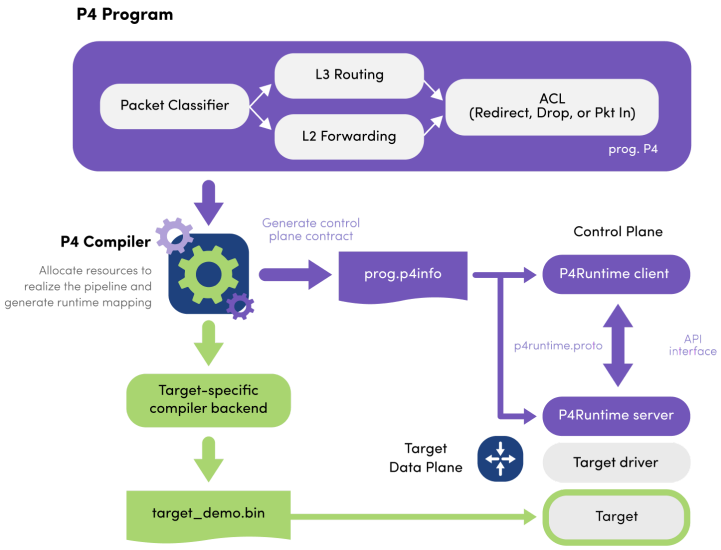

Figure 1.4 - P4 lifecycle

source: p4.org

The P4 lifecycle from the creation of the program file to the deployment of its compiled form in the target device can be conceptualized as shown in Figure 1.4, and adding to what was already described earlier, P4 programs and compilers are target-specific. The target can be hardware-based (field-programmable gate array (FPGA), Programmable application-specific integrated circuits (ASICs)) or software (running on x86). A P4 compiler generates the runtime mapping metadata to allow the control and data planes to communicate using P4Runtime. A P4 compiler also generates an executable for the target data plane (e.g., target_prog.bin), specifying the target device's header formats and corresponding actions.

2.2.2 P4Runtime[4]

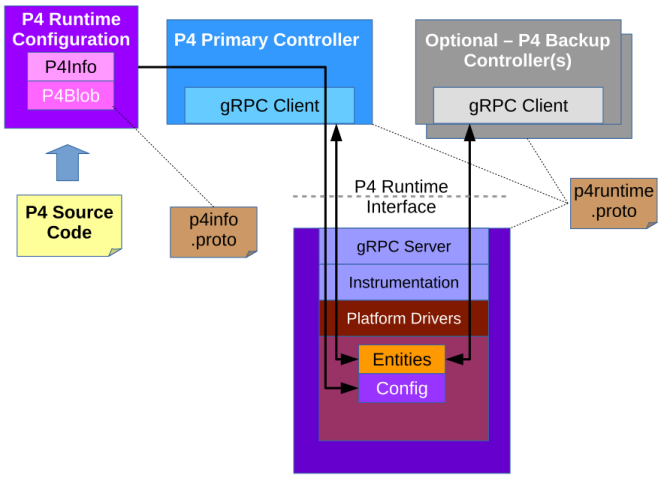

Figure 1.5 - P4Runtime Reference Architecture

Figure 1.5 represents the P4Runtime Reference Architecture. The P4Runtime API is a control plane specification for controlling the data plane elements of a device defined or described by a P4 program. P4Runtime only grants write access to a single primary controller for each read/write entity. A role defines a grouping of P4 entities. P4Runtime allows for a primary controller for each role, and a role-based client arbitration scheme ensures only one controller has write access to each read/write entity, or the pipeline config itself. Any controller may perform read access to any entity or the pipeline config. The API is specified

[4] P4Runtime - https://github.com/p4lang/p4runtime

by a Protobuf [5]file. The P4Runtime API is implemented by a program that runs a gRPC server which binds an implementation of an auto-generated P4Runtime Service interface. This program is called the "P4Runtime server." The server must listen on TCP port 9559 by default, which is the port that has been allocated by IANA for the P4Runtime service.

In the idealized workflow, a P4 source program is compiled to produce both a P4 device config and P4Info metadata. A P4Runtime controller chooses a configuration appropriate to a particular target and installs it via an RPC message. Metadata in the P4Info describes both the overall program itself as well as all entity instances derived from the P4 program — tables and extern instances. Each entity instance has an associated numeric ID assigned by the P4 compiler which serves as a concise "handle" used in API calls.

2.2.3 Grafana

Grafana[6] allows users to query, visualize, alert on, and understand metrics no matter where they are stored. Since the goal of the project is to conceive a dashboard for network management, a user needs to be able to visualize the network metrics in order to determine its health and make accurate decisions. Grafana gives the capability to easily instruct the tool in which data to query and how to display it in various types of interactive graphics and tables. It also allows the user to set alerts for certain metrics and browse, real-time observation of data collection, metrics timelines, and histograms.

2.2.4 Prometheus

Prometheus[7] collects and stores its metrics as time series data, i.e. metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels.

Metrics are numeric measurements. Time series means that changes are recorded over time. For this system, the metrics will be about traffic measured like the number of packages per second, the average size of these packages, and sizes of queues in the network devices, among others.

[5] Protobuf - https://protobuf.dev/
[6] Grafana - https://grafana.com/
[7] Prometheus - https://prometheus.io/

Prometheus works well for recording any purely numeric time series. It fits both machine-centric monitoring as well as monitoring of highly dynamic service-oriented architectures. In a world of microservices, its support for multi-dimensional data collection and querying is a particular strength.

2.2.5 Docker

Docker[8] is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.

These advantages are key when dealing with SDNs, where we want topologies to react fast and reliably to changes. With P4 being such a novel technology, Docker also provides us with stability during use which otherwise wouldn’t be possible.

[8] Docker - https://www.docker.com/

CHAPTER 3

SYSTEM REQUIREMENTS AND ARCHITECTURE

3.1 SYSTEM REQUIREMENTS

This section presents the system requirements specification, as a result of the first phase of the prototype development. It contains a description of the requirement elicitation process, a context description, some persona description, actors classification, use-case diagrams, non-functional requirements, and the system's assumptions and dependencies.

3.1.1 REQUIREMENTS ELICITATION

For the requirement, the elicitation process can be divided into a few stages. In the first stage, the more abstract goals set by the project presentation were specified. For that, in an interview with the coordinators, the team set about to understand the technologies needed to undergo the development of this project, as well as the conceptualization of real-world cases which could benefit from the use of this system.

In the next stage, and after acquiring some knowledge of the capabilities and limitations of the technologies, the team organized a few brainstorming sessions to idealize potential uses that could be requested from the system, the following cases were identified:

- Starting with small and simple, we conceived a scenario where the network was composed of four or five hosts, hosting stable and continuous services like the ones commonly found in simple web services (e.g. Apache[9], MySQL[10], ExpressJs[11]) and a single network device to connect them.
- In order to introduce a new set of devices, it was then conceived a network composed of more than one subnet, where one could be the most stable one (e.g. previous case) and the other with frequent disconnections and new connections from its hosts.
- While building on the previous example with the implementations of subnets, we conceptualize the implementations of virtual local area networks (VLANs), with the

[9] Apache - https://www.apache.org/
[10] MySQL - https://www.mysql.com/
[11] Express.js - https://expressjs.com/

added functionality of this configuration to be made dynamically. The use of multiple VLANs in a single subnet was also a subject of interest for experimentation.

- As the ultimate test implementation we devise a network where the topology needs to dynamically change to isolate a VLAN and/or a subnet when a specific source IP address is detected. This can then be expanded to create reactivity to other parameters and reactions.

3.1.2 CONTEXT DESCRIPTION

Since the system only accounts for one stakeholder, the network administrator, there are no differences in the behavior of the system in relation to its user. What can be accounted for, since the system allows for more than one user, is the different roles each can be attributed to and the authorizations each role possesses. There will be at least two main roles, administrator and observer, with other roles being user defined.

Firstly when deploying the system, the user will be set as the network administrator. This administrator role will be the only one capable of creating and destroying networks, activating new devices, and changing current network device configurations. Creating new user accounts and providing them with appropriate authorization will also be exclusive to this user.

After creating the account, the administrator can start by creating new devices one by one, or multiple at a time, which can then be organized into networks. The user can specify network attributes, like a name, the allowed number of hosts, and Dynamic Host Configuration Protocol (DHCP) pool ranges, among other common network attributes. Defining metrics is the next step, this can be done by selecting from a number of predefined metrics, or by providing P4 code.

Finally, after selecting the desired metrics, the user can then select which to be graphically displayed on the main page for visualization.

3.1.3 ACTORS

As said before, in all of the use cases, the network administrator will be the only actor, even if there could be multiple instances of it. This system's capabilities and interface are set to accommodate the needs and expertise of any professional with intermediate to advanced

technical knowledge of networking operations, service virtualization, and containerization. It is expected that this user can access the service from any common browser with access to the Internet and that no action is limited by the type of device from which the user accesses the dashboard.

3.1.4 USE CASES

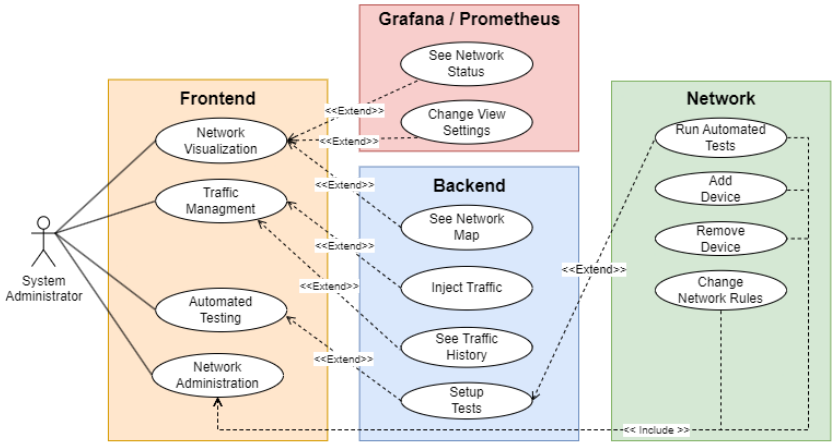

Figure 2.1 - Use case model

Figure 2.1 illustrates the global use case for the complete system, which can be divided into four main groups:

- **Frontend:** This relates to everything the user can see and direct access, it serves as the medium from which the user interacts with the system and from where it can visualize information relating to the system. Within this scope, some main uses were highlighted:
    - **Network Visualization:** Allows the user to view the current topology of the network. In list form or a drawn-out diagram, it displays all current network devices and their state information like name, addresses, and if currently up or down. Also includes the visualization of graphs illustrative of metrics currently being used.

      Priority: **High**.
    - **Traffic Management:** Includes, among other capabilities, the option to specify network traffic routing rules (e.g address to be blocked, subnets to be "shielded" from certain types of traffic). Some metrics to be implemented should be heavy hitter detection, the average packet size in transmission per

time unit, the cardinality of connections to hosts per time unit, and the cardinality of blocked or redirected traffic per time unit.
Priority: **High**.

- **Automated Testing:** For testing, if new rules imposed on the network are running correctly, one capability of the dashboard should be to inject “dummy” traffic and verify if the topology reacts accordingly.
  Priority: **Medium**.
- **Network Administration:** This encompasses all operations relating to the management of network devices. This could be the creation of new devices, the removal of these from the network, the enforcement of network traffic rules, creation of new subnets and VLANs. These can also comprehend the configuration of firewalls with allowed hosts in the network, prohibited traffic, or sectioning of devices.
  Priority: **High**.

- **Grafana and Prometheus:**
  - **See Network Status:** With Grafana, the user is capable of visualizing metrics graphically which are provided by Prometheus.
    Priority: **High**.
  - **Change View Settings:** In order to make the dashboard more convenient, the user will have the capability of choosing which graphics and layout he or she prefers to view.
    Priority: **Medium**.
- **Backend:** This section comprehends all services running in the background that support all user activity or automated services.
  - **See Network Map:** Feeding into the network visualization, it collects and treats the data that constructs the network map.
    Priority: **High**.
  - **Inject Traffic:** Is able to inject traffic directly into the network via temporary network devices.
    Priority: **High**.
  - **See Traffic History:** This communicates with the persistent database and allows the retrieval of the data relating to previous dates.
    Priority: **High**.

    - **Setup Tests:** Is able to run tests in the network that the user specified.
      Priority: **Medium**.
- **Network:** englobing devices and traffic running in the network.
    - **Run Automated Tests:** Comprehends all the dummy traffic and temporary network device configurations, along with the metrics collected from this traffic.
      Priority: **High**.
    - **Add Device:** Adding new devices to the network.
      Priority: **High**.
    - **Remove Device:** Removing devices to the network.
      Priority: **High**.
    - **Change Network Rules:** The network rules change will implicate a modification in the network's topology.
      Priority: **High**.

3.1.5 NON-FUNCTIONAL REQUIREMENTS

In this section a list of non-functional requirements is presented. These requirements specify what the system must provide in order to function in a satisfactory way for the user. These requisites can be divided into a few categories.

- **Availability:** These types of requirements are set in order to assure that the users of this system have as broad of access to it as possible.
    - For it all users must be able to connect with a personal and non-transmissible account that identifies them and the alterations each of them make to the network.
      Priority: **High**.
    - The system should provide an interface adaptable to multiple types of devices.
      Priority:
- **Usability:** this requirement describes the ease of use and intuitive action implemented in the system.
    - Every user account can be associated with multiple networks.
      Priority: **Medium**.

- Each network can be associated with multiple users (have multiple administrators).
  Priority: **Medium**.
- Each network should be capable of accommodating a variety of user-defined metrics. The creation of this metric should be simple (e.g approximate the way how a user creates e-mail filtering rules).
  Priority: **High**.

- **Security:**
  - All the information regarding a specific network should be limited to the users with access to it.
    Priority: **High**.
  - All of the information produced should be verified in order to prevent corruption of data and infiltration or exfiltration of it.
    Priority: **High**.
- **Reliability:**
  - Stability of the connection is paramount, it should be stable once established via the dashboard.
    Priority: **High**.
  - Packet drop should be minimized, and if it exists, it should be logged.
    Priority: **High**.
- **Capacity:**
  - The performance of the network, even with the metric collection, should not be impaired substantially.
    Priority: **High.**

3.1.6 ASSUMPTIONS AND DEPENDENCIES

This system is expected to be run in a Linux environment, with Docker and Node.js support and a graphical browser. During the system deployment, the user should have enough authorization to create and delete virtual network interfaces and create and destroy containers of virtual machines.

3.2 SYSTEM ARCHITECTURE

This section presents an overview of the system architecture, describing its domain model and underlying relationships, as its physical and technological model.

3.2.1 DOMAIN MODEL

In Figure 2.2 it can be observed the relationship between the different entities involved in the developed solution. Many of these entities were developed with the specific purpose of interacting with each other in this particular environment which does not allow for great modularity of each, which is not itself a downfall since it is intended for the solution to be deployed as a whole. First of all, the system possesses a central entity, the Service Broker which is responsible for mediating the distribution and storage of the produced data. Prometheus uses this broker to feed its time series data production which in turn will be used by *Grafana* to reproduce graphics so the end-user is able to easily visualize the metrics generated by the network devices. The P4 Runtime Controller API is one of the entities which is highly modular and has only required IP addresses in order to establish communication with BMv2[12] network devices (which are P4 capable virtual devices), and thus is agnostic to the environment in which these are deployed or how they are being used. The controller is also responsible for retrieving metrics generated by the devices via an HTTP server which can be queried by any external service. Because the system currently only works in a *Mininet-emulated* network, where multiple devices running BMv2 are deployed, a simple HTTP Mininet controller was developed in order to control the network without needing to rely on a CLI.

The authentication service has its own database where it stores the credentials and the users who are authorized to manage the service. The data that is generated and consumed is stored in a single *MongoDB* database located *off-site* in a remote virtual machine in *AWS*[13]. Finally, the website can be run on multiple devices with network capabilities and provides access to the service broker which in turn allows a user to manage the system remotely.

[12] Behavioral Model version 2 - https://github.com/p4lang/behavioral-model
[13] Amazon Web Services - https://aws.amazon.com/

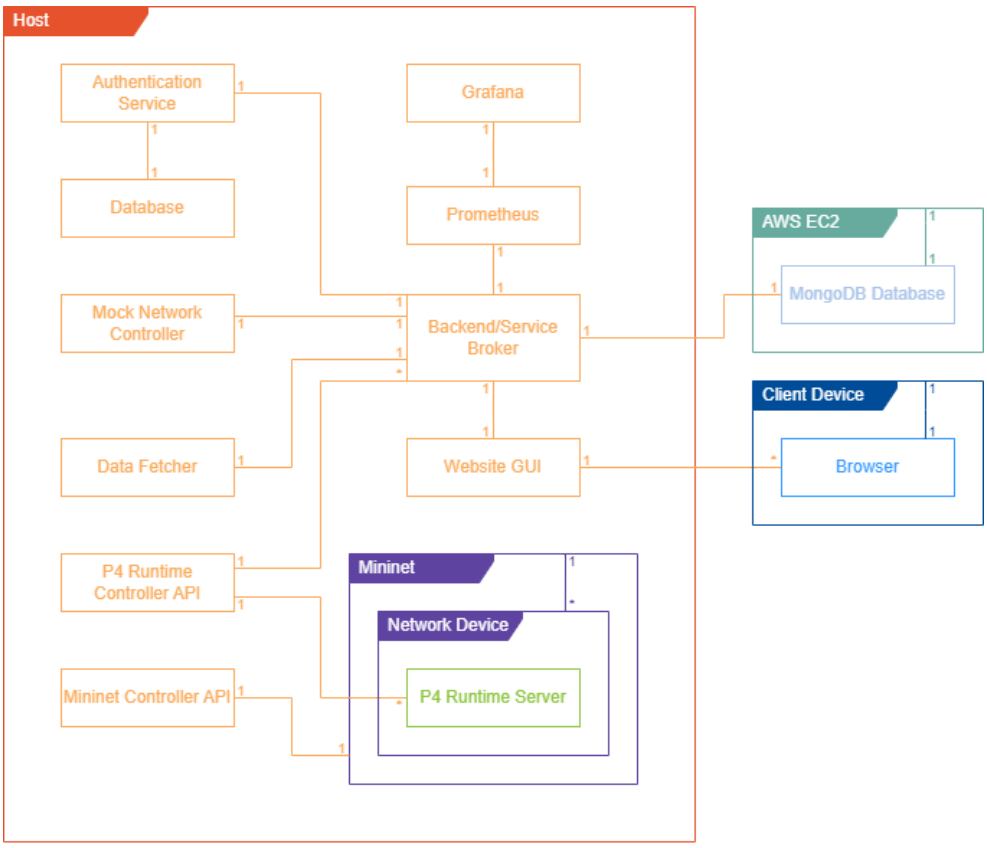

Figure 2.2 - Domain Model

3.2.2 PHYSICAL MODEL

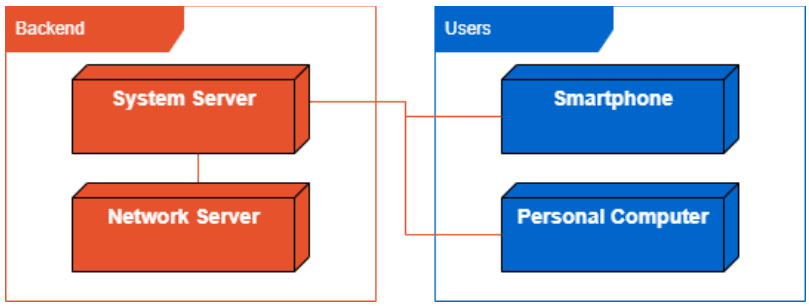

Figure 2.3 - Physical Model

Being this a system for a virtual service, the physical requirements for deployment are quite scarce. Like it is presented in Figure 2.3, there can only be three to four specified, on the user side, the two possible types of devices used by the user, and in the backend part of the service, all the system could be run from a single device or, like the diagram below, run the network and

management services in separate physical devices. The separation between system and network server is relevant because of the fact that a physical separation between the device that supports various system services (e.g databases, authentication services, etc) and one that possess the virtualized services and SDN components, is possible and can even be desired if a the SDN expand over large geographical areas.

3.2.3 TECHNOLOGICAL MODEL

As displayed in Figure 2.4, the adaptation to the technologies is done only at the server level. This system is expected to run on a Linux server, it will rely most heavily on Docker containers for most of its components, including Prometheus and Grafana deployed in containers also. The database of choice, being that most of the data are non-structured, is MongoDB, stored in AWS virtual machine for ease of utilization by multiple developers during the development of the system. For the website of the dashboard, to achieve good adaptation to multiple devices, ReactJs was chosen as it is simple and fast to develop, and by being mostly *client-side* rendered, lightens the load on the provider server. The P4 Runtime Controller is written in Python using Flask for HTTP request handling. It also utilized gRPC libraries in order to connect and communicate with the P4 Runtime Servers in the BMV2. The Mininet Controller API is similar to the previous service but instead of including libraries for gRPC, it includes Mininet's libraries in order to control the emulated network and its devices.

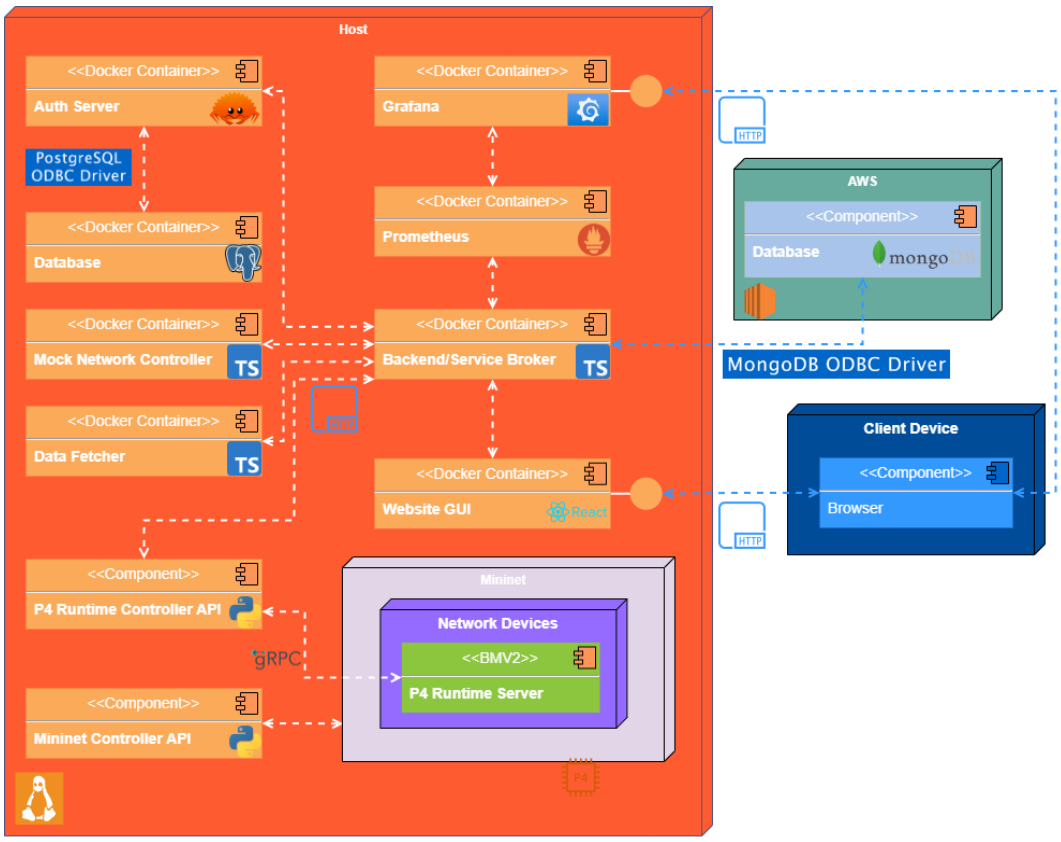

Figure 2.4 - Technological Model

CHAPTER 4

DEVELOPMENT AND IMPLEMENTATION

4.1 P4 DEVICES AND CONTROL

P4 devices are essentially software that behaves as a network device that can then be programmed via a compiled version of a P4 script. The most common and recommended of this type of software is Behavioral Model[14] Version 2 (BMV2) which comes in two versions: a simple switch implementation that can only be configured at the time when it is *spun up*, and a second version that exposes a gRCP[15] API denominated by P4Runtime API, for remote control operations. In the development of this project, only the second type is of interest since the goal is to be able to remotely program the devices.

4.1.1 P4Runtime REST API

Because P4Runtime only grants write access to a single primary controller for each read/write entity, it would not be a viable solution if multiple users may be connected and controlling one, or multiple, devices. To circumvent these limitations, and also to provide a layer of abstraction and compatibility for other services, a REST API has been developed. This allows for a single machine to be connected as the main controller, and at the same time, expose an API to access these controls.

Because P4Runtime is an RPC API, it requires complex message production in its requests The REST API also abstracts this by allowing the user to pass the desired parameters in POST requests, and the API will then construct the RPC messages accordingly. Also, for more complex but frequent requests, specific endpoints exist that do not require the user to specify parameters. Instead a simple GET request suffices in triggering the desired response from the device. This component is designed to be used by any service that allows for HTTP communication and is atomic in its behavior, not depending on any other service to run.

[14] Behavioral Model - https://github.com/p4lang/behavioral-model
[15] gRPC - https://grpc.io/

One of the more complex stages of communication with the devices is at the time of programming them with the compiled results of a P4 script. The compiler, and many other applications of P4, are undergoing development and are quite sensitive when it comes to dependencies, which makes it complicated to compile P4 scripts for a user whose system may not comply with all the requirements. To facilitate this process, as is visible in Figure 3.2, the REST API exposes an endpoint that abstracts all of this process by only requesting a source P4 script and identification of target devices to program, and it then deals with the compilations and programming of these in the background.

As of the time of this report construction, the REST API server is stateful, since from the time it is first initiated, it connects to the devices and keeps the connections alive for the duration of its life. However, it implements a graceful shutdown, by closing all gRPC connections before the service is terminated. Although stateful, all the requests, with the exception of those that deal directly with connecting and disconnecting from devices, made by users and propagated to the devices, do not change the state of the REST API itself, nor do requests with specific targets affect all other device's interaction with the REST API.

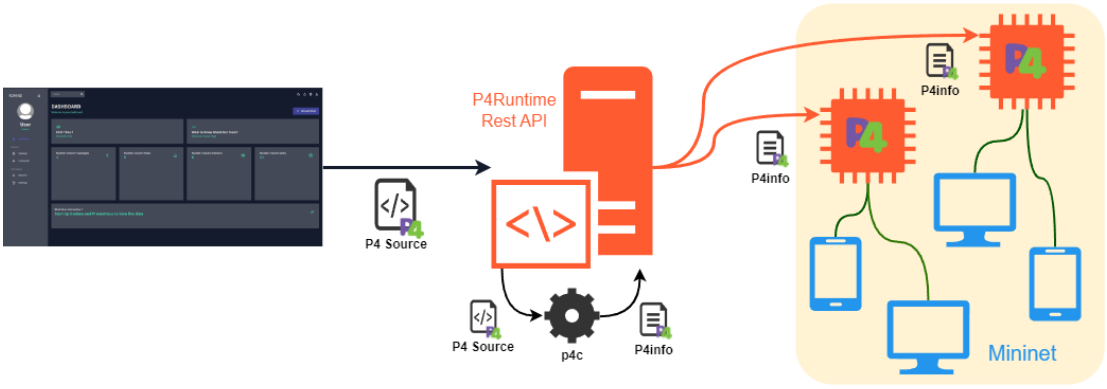

Figure 3.1 - Device programming using P4Runtime REST API

4.2 WEBSITE DASHBOARD

To ease the interaction with the system, a dashboard was created with the goal of being able to control devices with simple *click-actions* and visualize network topologies in a more intuitive way.

This *frontend,* has several functionalities, such as creating tables to format information relative to the network's topology, such as hosts, switches, and links, allowing the user to

customize and organize the data according to their preferences. In addition, it provides customizable visualizations of the different available networks based on the user's preferences. It also provides useful information to assist new users and the team responsible for the project's development. Finally, it allows the user to send various commands to the network, enabling quick and efficient execution of operations and data retrieval. A view of the main dashboard can be viewed in Figure 3.2.1.

The *frontend* was developed using several technologies, including.

- ReactJS[16]: a JavaScript library used as the foundation for the *frontend*, providing a solid and efficient structure.
- MUI[17]: a set of React add-ons used to format data and assist in the visualization of the *frontend*.
- JavaScript[18]: used to create requests and connect the *frontend* with the rest of the project.
- CSS[19]: used to add additional styles to the project that are not integrated into the MUI addons.

Although providing a solid foundation, the implemented *frontend* can still be improved. One of the possible improvements is the creation and availability of new commands, allowing the user to customize the network according to their preferences. Additionally, it would be beneficial to invest in performance and usability enhancements, improving response speed and the user experience during interaction with the *frontend*.

[16] ReactJS - https://react.dev/
[17] MUI - https://mui.com/
[18] JavaScript - https://www.javascript.com/
[19] CSS - https://www.w3.org/TR/CSS/#css

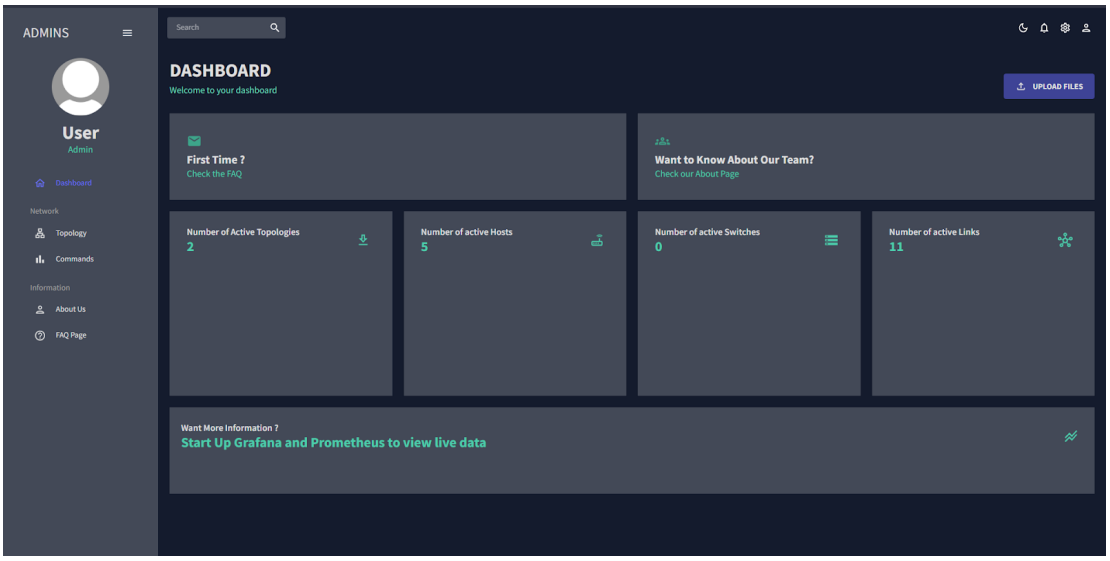

Figure 3.2 - Main dashboard

4.3 INFRASTRUCTURE

The backend of the project was developed exclusively based on the microservices architecture model. The primary driving force behind selecting this architecture, as well as the technologies employed, stemmed from a strong inclination toward acquiring knowledge and proficiency in emerging technologies. Consequently, it is worth noting that the solutions chosen may not always align optimally with the specific contextual requirements, as that was not the sole criterion guiding the selection process.

4.3.1 SERVICES

- Nginx [20]- Used as a reverse proxy between docker containers and the P4Runtime REST API.
- Mock Network Controller - Engine to simulate data produced by the P4Runtime REST API, useful during testing phases.
- Authentication/Authorization - A service implemented with Rust[21] and connected to a PostgreSQL[22] database, which is responsible for authentication and authorization.

[20] Nginx - https://www.nginx.com/
[21] Rust - https://www.rust-lang.org/
[22] PostgreSQL - https://www.postgresql.org/

- Data Provider - A service developed using TypeScript[23] and connected to a MongoDB[24] database, which is responsible for feeding Prometheus, which in turn feeds Grafana.
- Network Data Fetcher - A service implemented in C#[25] and connected to a Redis[26] message queue, responsible for collecting network data and propagating it to the other services.
- Alarm Scanner - A service developed in C# and connected to a Redis message queue and a PostgreSQL database with alarm configurations. It is responsible for filtering relevant network data for the alarms.
- Alarm Processor - A service developed in C# and connected to a Redis message queue and a PostgreSQL database with alarm configurations. Its responsibility is to validate if the criteria for triggering an alarm are met.
- Email Service - A service developed in C# and connected to a Redis message queue. Its responsibility is to notify users of the occurrence of an alarm.
- Alarm Configuration API - A service developed in C# and connected to a PostgreSQL database. Its responsibility is to receive new alarm configurations and store them.

[23] TypeScript - https://www.typescriptlang.org/
[24] MongoDB - https://www.mongodb.com/
[25] C# - https://learn.microsoft.com/en-us/dotnet/csharp/
[26] Redis - https://redis.io/

CHAPTER 5

FINAL NOTES AND FUTURE WORK

The development of this project was an intensive yet interesting and motivating introduction to the world of software defined networks and particularly to P4 language.

The capability of separating the control and data planes, combined with the capability for a flexible configuration of the first, makes P4 an extremely powerful technology and an enticing platform on top of which much more complex systems can be designed and implemented.

The team considers that the goal for this project, the implementation of a monitoring system using P4 was accomplished, which in turn validated the hypothesis of developing such systems. However, it is still a desire of the team to dive deeper into a more robust and capable solution based on the work already presented in this report.

Left to attain, was the goal of implementing some autonomous reactivity, based on real-time analysis of the monitored data, which in turn could be an interesting field to deepen with machine learning, where the system would "*learn*" about how to deal with devices configurations according to the received metrics and the desired state of the network.

Also, the P4Runtime REST API presents itself as a strong contender for a much larger application, if its capabilities are expanded and perfected, which can be easily achieved.

Having said this, the team plans to continue developing some of the software created in this project, as an open-source project (OSS) with its organization in GitHub. Because of that, it was decided to officialize the identity of this application, with the name of P4Sentry.

Currently, steps are being made to leave the emulated environment of Mininet in turn for a real environment composed of Docker containers, virtual machines, or any other types of services; by developing a BMV2 Docker image with BMV2 at its core.

Finally, it is important to enphasize that all of the project was develop under an Open Up and Agile methodology. From the start, coordination between members and taks to be developed was done utilizing GitHub Projects, with the capabilities of a Kanban board, in order to prioritize fast and complete issue resolution over volume of issues, as we can see in Figure 4.1

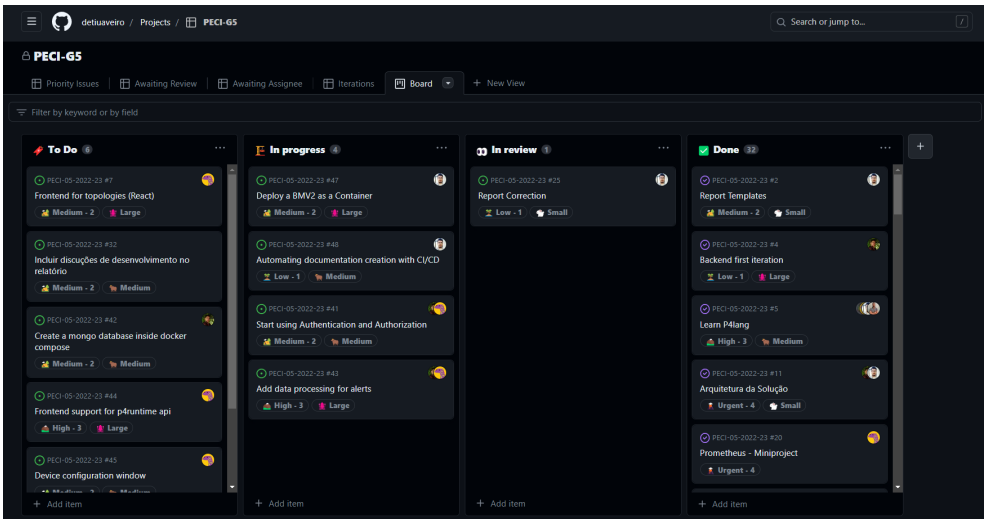

Figure 4.1 - GitHub Projects Kanban board

This tool was indispensable for the smooth development process. At the beginning of every iteration, of which there were four, an initial set of new issues declaring necessary feature to be implemented was declared.

Also, to ease and avoid disruption between atomic components, every one of these was developed under its own branch of the repository was it is visible in Figure 4.2.

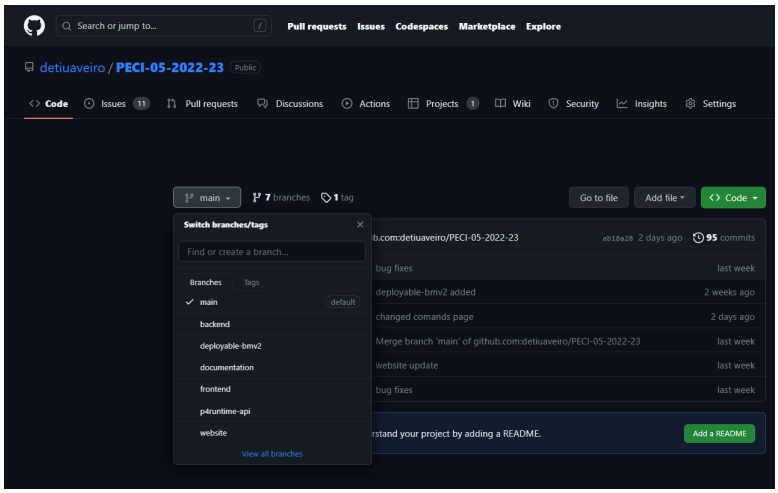

Figure 4.2 - GitHub repository branch division

Review processes were also implemented. Has each of the members was tasked with developing a separate component, at the time of combining them, all of the member would,

together, review and analysis the system of each one and detect and fix possible *bugs* befores merging branches.

REFERENCES

REFERENCES

[1] P4.org. “Improving Network Monitoring and Management with Programmable Data Planes.” Open Networking Foundation, Open Networking Foundation, 25 Sept. 2015, https://opennetworking.org/news-and-events/blog/improving-network-monitoring-and-management-with-programmable-data-planes/.

[2] Lee, Jeongkeun JK, and Mukesh Hira. “Inband Network Telemetry (INT): History, Impact and Future Direction.” P4 2022 Workshop. 24 May 2022.

[3] P4.org, “In-Band Network Telemetry (INT).” In-Band Network Telemetry (INT) Dataplane Specification, 2.1, P4.Org, 11 Nov. 2020, https://p4.org/p4-spec/docs/INT_v2_1.pdf. Accessed 10 Dec. 2022.

[4] Christos, Demertzis. “Implementation of Network Telemetry System With P4 Programming Language.” Department of Applied Informatics, University of Macedonia, University of Macedonia, 2020, pp. 1–75.

[5] Junjie Geng, Jinyao Yan, Yangbiao Ren, and Yuan Zhang. 2018. Design and Implementation of Network Monitoring and Scheduling Architecture Based on P4. In Proceedings of the 2nd International Conference on Computer Science and Application Engineering (CSAE '18). Association for Computing Machinery, New York, NY, USA, Article 182, 1–6. https://doi.org/10.1145/3207677.3278059