universidade de aveiro  deti departamento de eletrónica, telecomunicações e informática

SOFTWARE DEFINED NETWORKS MONITORING SYSTEM

PROJECT IN COMPUTER AND INFORMATICS ENGINEERING

AUTHORS

85085 Diogo Dias

88964 Afonso Cardoso

89119 João Machado

93444 David Araújo

98391 Vasco Santos

103574 Guilherme Craveiro

SUPERVISORS

Prof. Daniel Corujo

Prof. José Quevedo

David Santos

Rui Miguel Silva

DECEMBER 2022

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 at a faster speed. However, flexibility has been a largely overlooked issue when it comes to the evolution of the networks that are present in all the systems that run our lives.

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), as we used to look as constants like the network growth meaning the installation of new hardware like switches and routers, out exchanging the existing ones with superior models with more physical ports, the configuration of these devices has been done with a few command in the command line interface (CLI), ends. What could networks be if, like many other aspects in the evolution, so many of these devices be substituted 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 and automate network management and reactive/dynamic topologies in a way that was previously not possible.

The proof of concept is the implementation of a monitoring system that, by taking advantage of all the actors being virtualized, is capable of interacting with them in an automated fashion to collect traffic flow data and possibly reconfigure them in real-time in a reactive fashion to the data collected.

ACKNOWLEDGMENTS

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

5G	5th Generation Mobile Network
API	Application programming interface
ASIC	Application-specific integrated circuit
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
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
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 new faster, and more capable devices to 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, the question poses: 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 on 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 comes has a considerable improvement over the existing infrastructure, the issue of configuration and reconfiguration still is not solved has SDNs focus on the creation of new virtual devices, but once these are deployed, unless they are “cloned” from other devices already containing the intended configuration, these will need to be configured and re-configured every time the topologies need to change.

To solve the problem of configuration and reactiveness of devices to the traffic flow over time, solutions like P4 (Programming Protocol-independent Packet Processors) have appeared. P4 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.

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 Informatic 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, 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 the dashboard, the user will be able to create and manage new/existing network devices, redefining its 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 four main parts: dashboard creation, P4 capable devices, controller, header, and metrics specification. Then, chapter 5 presents the two usability tests conducted during this thesis,

including its sample, method, and results evaluation. At last, chapter 6 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 network device by also allowing the design and creation of custom headers an "packets". This direct interaction with the pipeline by not only observing but altering the traffic itself opens ups 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.

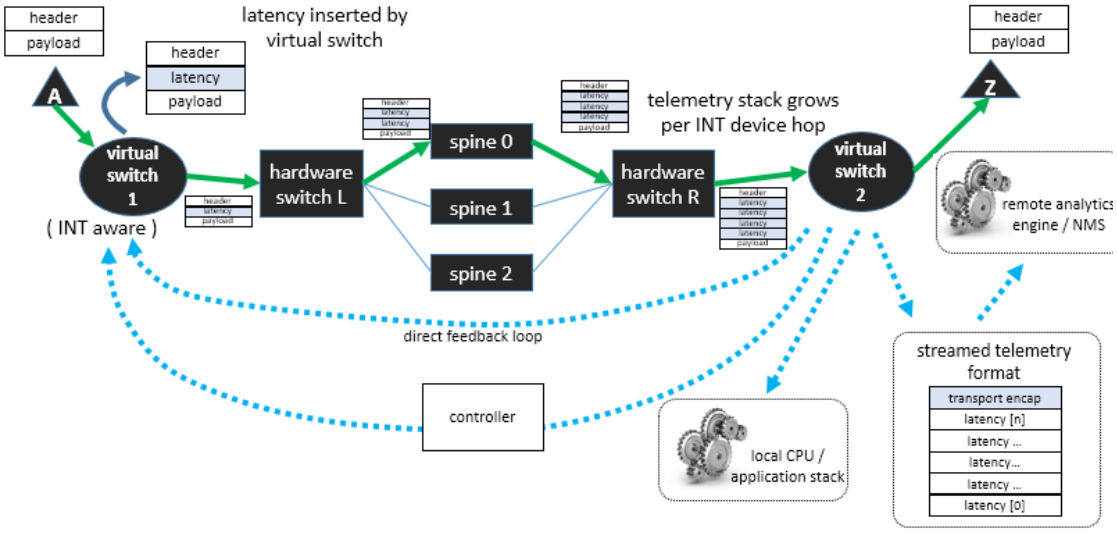

Figure 1.1 - Measuring and reporting end-to-end latency between virtual switches

source: [1]

2.1.2 Inband Network Telemetry (INT): 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 velocity 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.

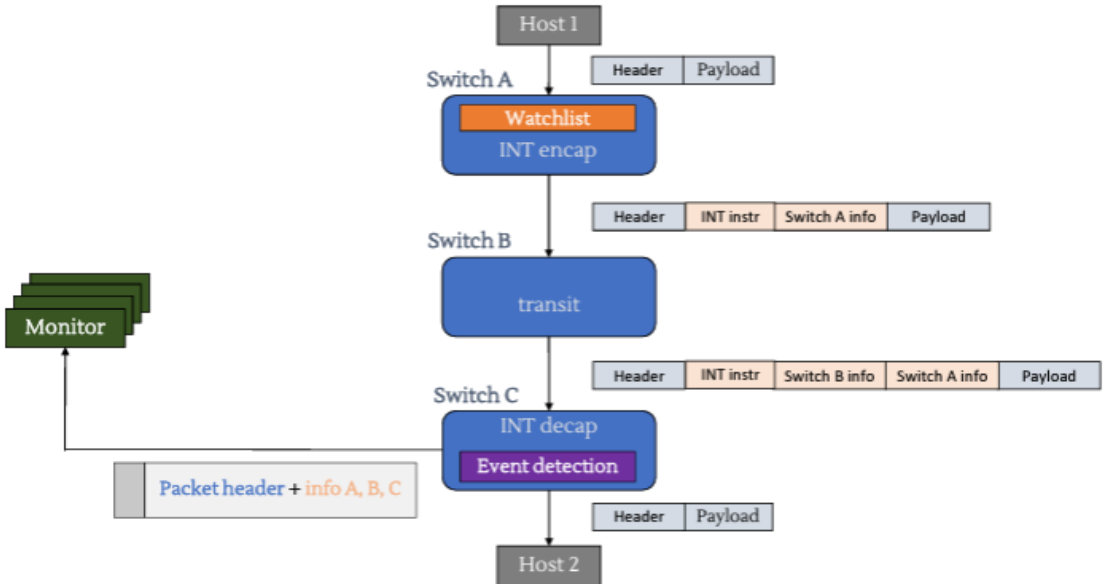

Figure 1.2 - Initial concepts of INT integration in frames and its role in telemetry collection and transport

source: [2]

2.1.3 In-band Network Telemetry (INT) 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[1] 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.

[1] Mininet - www.mininet.org

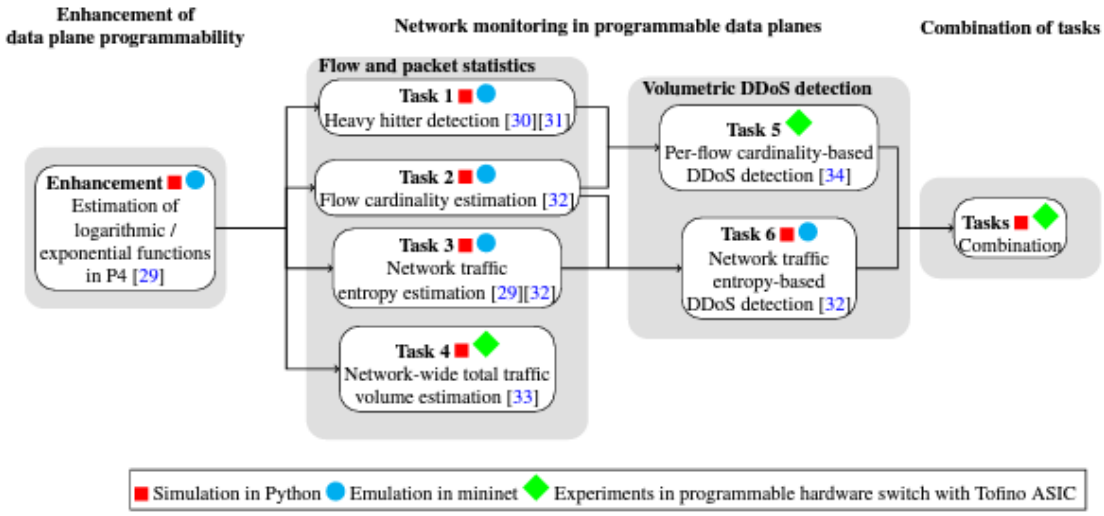

Figure 1.3 - Scheme of the thesis

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.

2.2.1 P4

Programming Protocol-independent Packet Processors (P4) is a domain-specific language for network devices, specifying how data plane devices (switches, NICs, routers, filters, etc.) process packets. Before P4, vendors had total control over the functionality supported in the network. And since networking silicon determined much of the possible behavior, silicon vendors controlled the rollout of new features (e.g., Virtual Extensible Local Area Network (VXLAN)), and rollouts took years. P4 aims to flexibilize this by allowing application developers and network engineers to implement specific behavior in the network, and changes can be made in minutes instead of years.

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 (target_prog.bin), specifying the target device's

header formats and corresponding actions. 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:

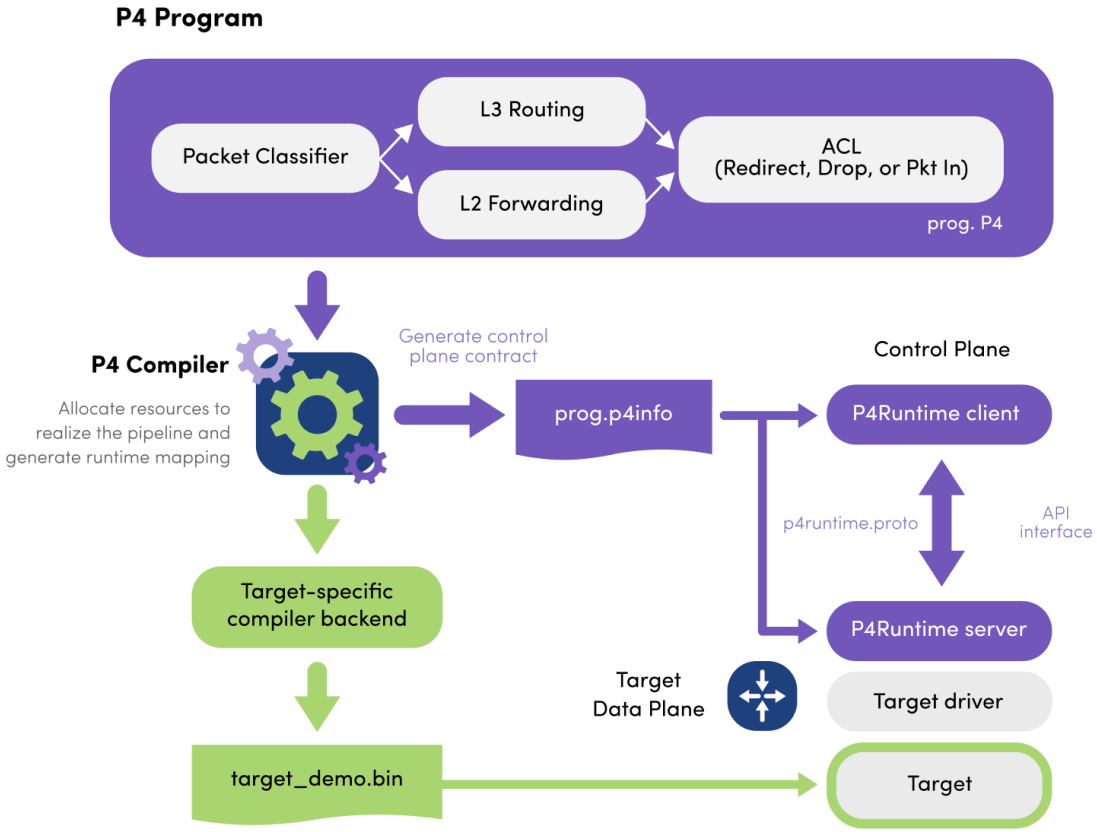

Figure 1.4 - P4 lifecycle

source: p4.org

2.2.2 Grafana & Grafana Mimir

Grafana[2] 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] Grafana - https://grafana.com/

Grafana Mimir is an open-source, horizontally scalable, highly available, multi-tenant, long-term storage for Prometheus[3].

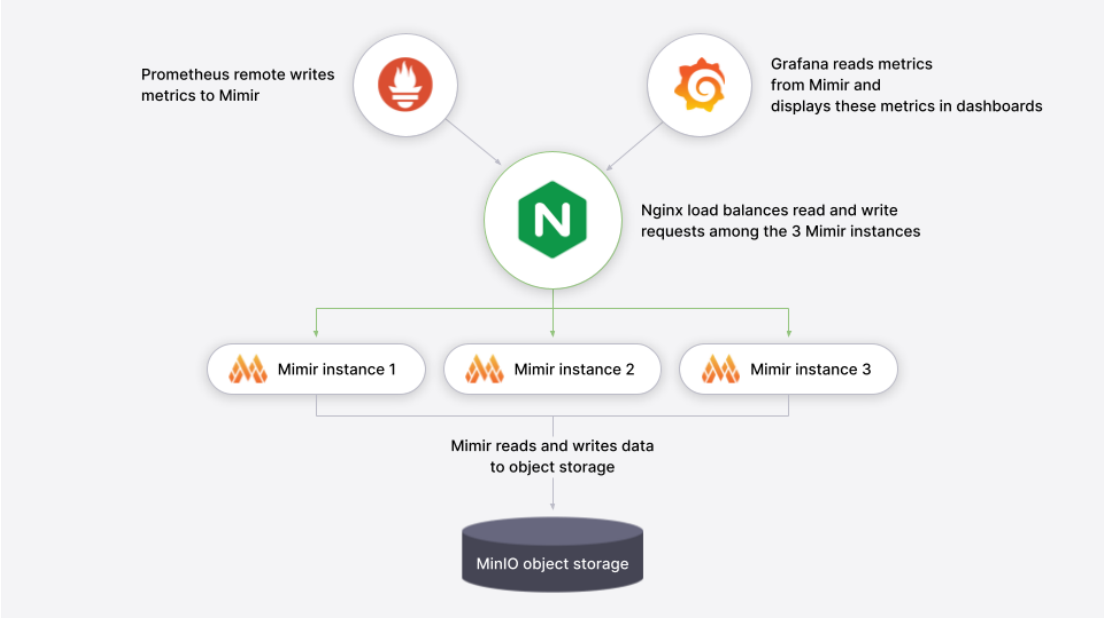

Figure 1.5 - Architecture from Mimir integration

source: grafana.com

As seen in Figure 1.5, Grafana Mimir is optimal for the collection of metrics in our solution since it does not require any special modification to the data flow by interacting as a middleman between Prometheus and Grafana

2.2.3 Prometheus

Prometheus 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.

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

[3] Prometheus - https://prometheus.io/

architectures. In a world of microservices, its support for multi-dimensional data collection and querying is a particular strength.

2.2.4 Docker

Docker[4] 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.

[4] 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[5], MySQL[6], ExpressJs[7]) and a single network device to connect them.
- In order to introduce a new set of devices, it was then conceived a network composed by more than one subnet, and where one could be a 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

[5] Apache - https://www.apache.org/
[6] MySQL - https://www.mysql.com/
[7] 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

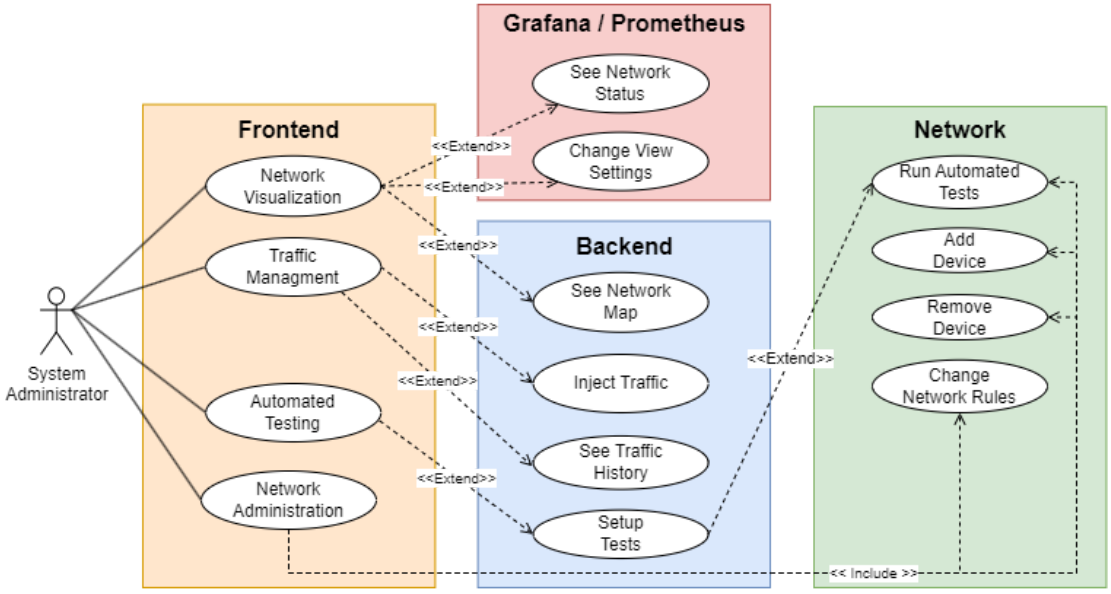

Figure 2.1 - Use case model

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 2.3 we can see the same entities as in the previous diagram but with the specifications of the relations between them. To begin with, there is a user entity that will interact with a single Web GUI, which itself can be used by multiple users. This entity will be responsible for displaying the different views of the dashboard to said users. The Web GUI will interact with three entities, the controller, Grafana, and Prometheus. The Grafana entity is responsible for generating graphics and can do so to multiple dashboards, being that it will have a single instance, it collects the data for graphic generation from multiple Mimir instances which store the data, serialized by Prometheus, in multiple instances. Prometheus by receiving data from the Data Analyser serializes it, and logs it, making it available for use by Mimir or directly to the dashboard. The Data Analyser is responsible for calculating the metrics set by the user in the dashboard, for this it utilizes the data received by the Controller. The Controller is responsible for communication with the different network devices, in networks of considerable size there can be multiple controllers, each communicating with a set of network devices. The Controler communicates these devices' status to the dashboard and when a new configuration is submitted, it will either create a new P4 file, which will be then compiled and pushed to a device or communicate directly with a device via the P4 Runtime API to apply the said configuration.

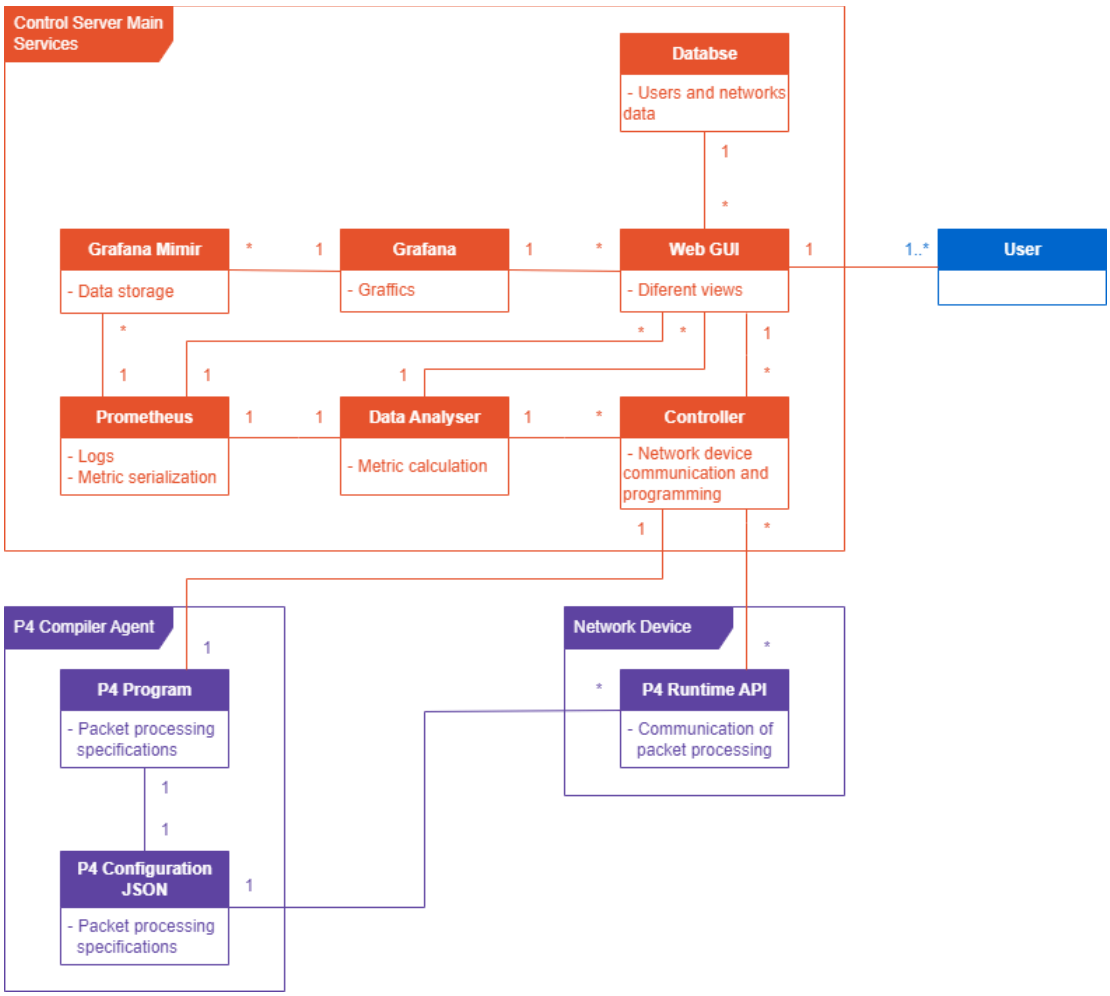

Figure 2.2 - Domain Model

3.2.2 PHYSICAL MODEL

Being this a system for a virtual service, the physical requirements for deployment are quite scarce. Like it is presented in Figure 2.5, 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.

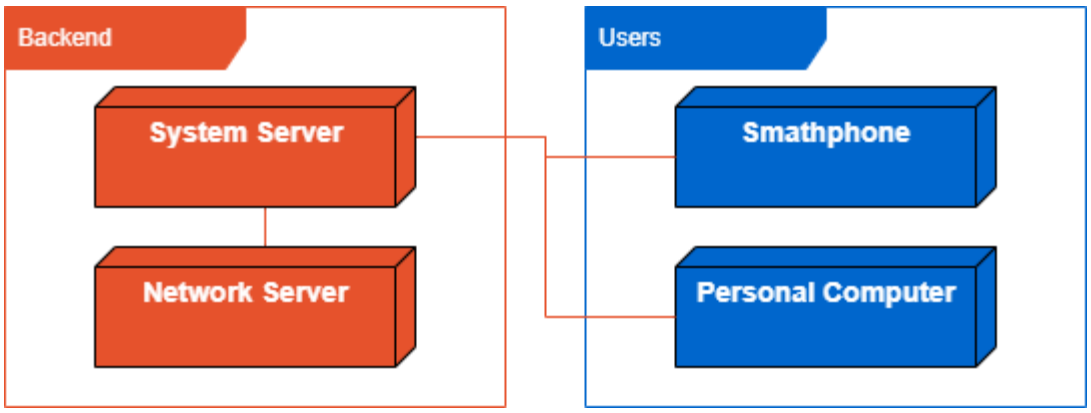

Figure 2.3 - Physical Model

3.2.3 TECHNOLOGICAL MODEL

As displayed in Figure 2.5, the adaptation to the technologies is done only at the server level. As said before, 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 is non-structured, is MongoDB. For the website of the dashboard, to achieve good adaptation to multiple devices, ReactJs was chosen as it is simple and fast to develop.

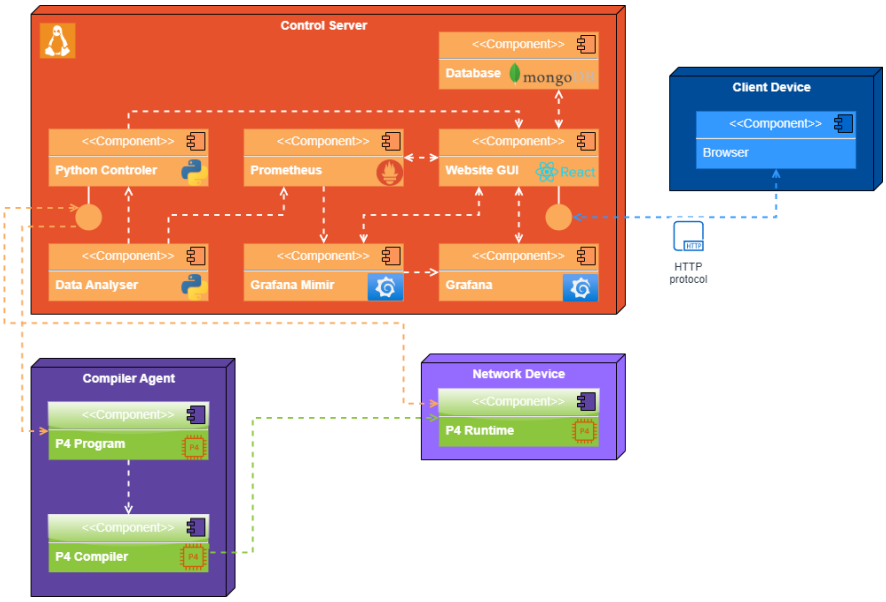

Figure 2.4 - Technological Model

CHAPTER 4

PRELIMINARY TESTING

As previously mentioned, this solution relies heavily on two main concepts: the ability to monitor traffic by direct interaction with the frames in flow, and the ability of remote, real-time, and reactive reconfiguration of network devices.

The first test had the objective of comprehending how P4 allows a user to configure network devices and how this affects traffic manipulation, and the second test had the objective of comprehending how, by using Pyhton *p4runtime_lib* which implements gRPC, it is possible to reconfigure devices.

4.1 P4 Devices Networks with p4app

P4app is a docker container created by the developers of P4, which allows quick and easy testing of simple P4 code. One can use this container by passing a directory containing configurations for a Mininet topology, P4 code for the configuration of devices, and/or Python code for the configuration of hosts it using a Mininet-created one is not desirable.

P4app after binding this directory within an internal one, what it does is using these files to construct what could be compared with a self-contained, “black-box styled”, simulation inside the docker. After running this “simulation” the docker container will return some specific files like the standard output of each existing host containing what that host has printed to the console during the simulation, the standard output of a controller (if one was created) and, if requested in the simulation start, a Packet Capture (PCAP) file containing all the traffic that occurred during the simulation. These files can then be used to analyze and extrapolate how the traffic has been managed and if the P4-configured device acted as intended.

With these capabilities, what was tested was a simple implementation of a router in a two-network setup with each network having a single host, this allowed us to comprehend how P4 is capable of specifying the treatment made to each arriving and departing frame, this includes the simple operations of changing source and destination MAC addresses and how the

detection to packet sender and the recipient is made by means of Longest Prefix Match (LPM) with the IP addresses.

4.2 Controllers

The other major advantage of these systems, is the remote reconfiguration of the devices, to do so the user relies on what are called *controllers*. These controllers are nothing more than programs (in Python, Java, C++, or others) that via gRPC interact with the devices by implementing specific API which allows them to enforce rules, in the case of P4 most of the time, for packet field and desired value matching, which can then trigger a function (in P4 called *action*) with which a certain packet modification will be applied. This is quite interesting since this allows the user to configure the device to act in the manipulation of the packet according to any desired value in any desired field of the received frame.

For this testing, it was used a simple controller which enforced the rule that any packet with a source IP address specified by the user in the Python code, would be reflected back to the sender, but any other packet with a different IP source address would transit unphased by this action. This test was quite useful in order to visualize the impact that real-time configurations in the network devices had on the traffic flow.

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