Software Defined Networks Monitoring System

Authors

David José Araújo Ferreira
João Tomás Pires Machado
Guilherme João Dos Santos Craveiro

Orientation

Professor Daniel Corujo
Professor José Quevedo
David Santos
Rui Miguel Silva

Computer and Informatics' Engineering Project, 3º year, MIECT/LECI

June 2023

P4SENTRY

universidade de aveiro
theoria poiesis praxis

deti departamento de electrónica, telecomunicações e informática

Objective

This project is focused on virtualized networks, from hosts to network devices, and how that allows us to monitor and control them as a whole. In Software Defined Networks, monitoring the connections is particularly important as it can be a way of ensuring stability in a host machine or stability of a network if they do not share the same physical space.

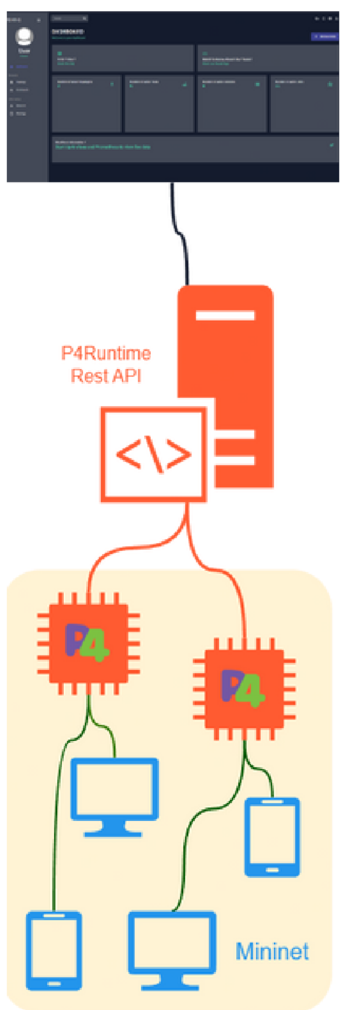
With the goal of developing a monitoring system that is capable of consulting network devices themselves and retrieving metrics specified by the user. This system needs to possess a dashboard for easy interaction regarding the reconfiguration of the device's settings and visualization of the metrics collected.

Environment

gRPC Flask

- **P4 Language** - 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.
- **P4Runtime Rest API** - Developed by our team, a Flask API that makes it possible to abstract the gRPC communication with the devices, to HTTP communication, and much more.
- **Docker** - Container networks are one of the main stages of SDNs, being able to deploy containers as network devices make it possible to control these networks.
- **Prometheus & Grafana** - Receive and generate time-series data streams for graphism generation.

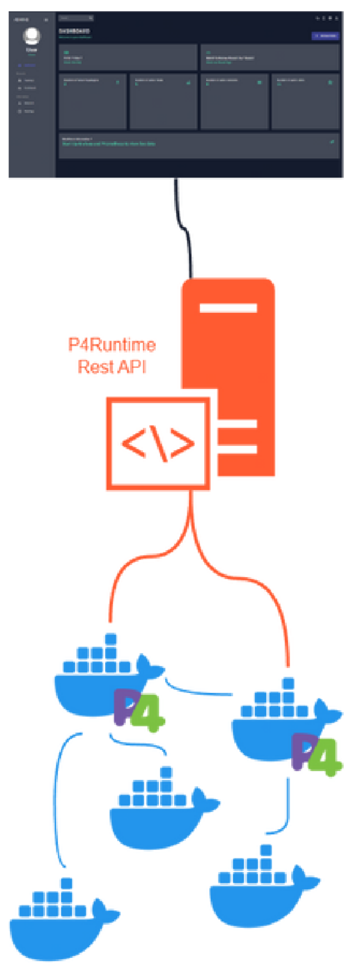
Architecture



Architecture of P4Sentry using a Mininet emulated network

P4Sentry is but a construction of various **atomic components**. From the beginning of development, each one of the components was designed to be self-contained and **capable of being used by itself** or in conjunctions with others. These atomic components are envisioned to work in the following manner:

- **P4Runtime REST API** - While gRPC is used by default by each P4-capable device to connect to a controller, it only allows for a single connection at a time. By running an **intermediary API**, this can maintain a connection to each device and multiple users can send requests to it in order to control one or **multiple devices** at a time. Being HTTP REST, it also offers **more compatibility** with other services and makes it possible to expand functionalities like ***in-request*** **P4 compilation**.
- **Dashboard** - Built using React, it allows a user to visualize the data collected in real-time, giving an overview of the state of the network. It also allows the user to see a **graphic representation of the network** and the connection between the nodes. It uses the P4Runtime REST API to connect to nodes and retrieve information from them. It then connects to *data fetchers/processors*, built with Typescript, that "consume" the data and "*feed*" it to Prometheus which will then create time-series streams for Grafana.
- **Mininet** - Currently the P4Sentry is still in a *proof-of-concept* state, as it operates entirely in an **emulated network using Mininet**.



Architecture of P4Sentry with a Docker network

With it, it is possible to emulate most aspects of a network, along with emulated host devices that can simulate traffic using *iperf3* or *pinging*.

The P4 language gives the user close to complete control over the processing of packets in the network. The **user can write its own P4 scripts** which are then deployed to the devices. However, during this project, it became evident the **difficulty of developing in, and for P4 environments and devices**. The software, although recent in existence, seems to **lack support** and **comprehensible documentation**, and the language itself has a **steep learning curve** due to the complexity of its **syntax**.

The software support for compilers and libraries was also one of the main reasons that **lead to the development of the P4Runtime REST API.** By keeping a single system to deal with all of the procedures involving P4, it relieves the user of the cumbersome process of frequently failed compilations or missing dependencies.

Currently, efforts are being made to leave the emulated network in turn for a real software defined one, for that, a **P4-capable network device is needed** in a way that it could be **easily deployed**, if and when needed, or shutdown if the network no longer needs it. This is the goal with the **BMV2Watchdog** Docker image. An image that when deployed will create a container that can then **act as a virtual network device**, dealing with network operations between containers and/or other running services.

Final Thoughts

P4Senty GitHub Organization

Understanding monitoring of SDNs is still very much a challenge, and implementing it in real systems even more so. The technology making it possible is not yet mature, but it shows promise and, most of all, flexibility for developing diverse and complex systems. P4Sentry began with this project, but it will continue in development as an open source project, with the main focus on BMV2Watchdog and increase dynamism and reactivity to traffic monitored health.

References

OpenFlow - https://opennetworking.org/
Mininet - www.mininet.org
Grafana - https://grafana.com/
Prometheus - https://prometheus.io/
Docker - https://www.docker.com/
P4 Language - https://p4.org/
P4Sentry - https://github.com/P4Sentry

Project's website