UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MATHEUS SAUERESSIG

# A Framework for Behavioral Fingerprinting in Programmable Networks

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Lisandro Zambenedetti
Granville
Coadvisor: Dr. Muriel Figueredo Franco

Porto Alegre
February 2024

*“The world is afflicted by death and decay.*

*But the wise do not grieve, having realized the nature of the world.”*

— Buddha in Sutta Nipata

## ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Professor Lisandro Zambenedetti Granville for providing both present and future opportunities in the exciting field of Computer Networking. Additionally, I extend my appreciation to Professor Muriel Figueredo Franco for his reliability, support, and assistance, this work owes its flourishing to his invaluable contributions. Heartfelt thanks go to my mother and father for their unwavering support, both emotionally and materially. To my friends, I am grateful for your enduring support, no matter the circumstances. I love each and every one of you.

Special appreciation goes to Dr. Ricardo Fasolo for assisting me during this challenging time, and I am grateful to my friend Julia for introducing me to him.

I extend my thanks to every member of the Universidade Federal do Rio Grande do Sul, including all workers, professors. Each contribution from these individuals has played a crucial role in shaping my education and identity. I am indebted to everyone who has cared for me, as their acts of kindness have paved the way for my journey to this point.

I also express gratitude to Shakyamuni Buddha for illuminating the path to bliss and truth and for teaching the importance of leading a moderate life. Recognizing our interconnectedness with life, death, and the entire Universe, let us embrace the joy of being part of something greater than the individual "I" and release our mundane perspectives of the world.

## ABSTRACT

The evolving computer network landscape has enabled programmability in various network aspects, including Software-defined Networking (SDN) for control plane programmability and the introduction of the Programming Protocol-independent Packet Processors (P4). P4, a vendor-independent protocol, allows programmability on the data plane, offering flexibility for new services and applications. However, this flexibility introduces the need for automated solutions to monitor and manage the security of evolving networks and services. In this work, we propose FEVER, a framework utilizing P4-based telemetry and network device (switch) resource consumption to create fingerprints of network and P4 application behaviors. FEVER provides a comprehensive approach to identifying network anomalies through various metrics. The framework was evaluated in a virtualized scenario using unsupervised Machine Learning (ML) algorithms to detect diverse P4 program behaviors and traffic overload, demonstrating its potential for early detection of malicious activities in programmable networks. The results indicate high accuracy in identifying misbehavior and detecting sudden changes in P4 programs affecting the network.

**Keywords:** Programmable Networks. P4. Switch. Computer Networks. Behavior Fingerprint.

**Uma Abordagem para Identificação de Comportamentos em Redes Programáveis**

## RESUMO

A evolução do cenário de redes de computadores possibilitou a programabilidade em diversos aspectos de redes, incluindo Redes Definidas por Software (SDN) para programabilidade do plano de controle e a introdução dos Processadores de Pacotes Independentes de Protocolo Programável (P4). O P4, um protocolo independente de fornecedor, permite a programabilidade no plano de dados, oferecendo flexibilidade para novos serviços e aplicações. No entanto, essa flexibilidade introduz a necessidade de soluções automatizadas para monitorar e gerenciar a segurança de redes e serviços em evolução. Neste trabalho, propomos o *FEVER*, um framework que utiliza telemetria baseada em P4 e o consumo de recursos de dispositivos de rede (*e.g.*, switch) para criar impressões digitais do comportamento da rede e de aplicações P4. O FEVER oferece uma abordagem abrangente para identificar anomalias de rede por meio de várias métricas. O framework foi avaliado em um cenário virtualizado usando algoritmos de Aprendizado de Máquina (ML) não supervisionados para detectar diversos comportamentos de programas P4 e sobrecarga de tráfego, demonstrando seu potencial para a detecção precoce de atividades maliciosas em redes programáveis. Os resultados indicam alta precisão na identificação de comportamentos inadequados e na detecção de mudanças repentinas nos programas P4 que afetam a rede.

**Palavras-chave:** Redes Programáveis, P4, Switch, Impressão Digital, Detecção de Anomalias.

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| **ABNT** | Associação Brasileira de Normas Técnicas |
| **QoE** | Quality of Experience |
| **QoS** | Quality of Service |
| **ML** | Machine Learning |
| **NN** | Neural Networks |
| **OCSVM** | One-Class Support Vector Machine |
| **LOF** | Local Outlier Factor |
| **RF** | Random Forest |
| **CPU** | Central Processing Unit |
| **RSS** | Resident Set Size |
| **RAM** | Random Access Memory |
| **KB** | Kilobytes |
| **OS** | Operating System |
| **PID** | Process Identifier |
| **TCP** | Transmission Control Protocol |
| **UDP** | User Datagram Protocol |
| **SCTP** | Stream Control Transmission Protocol |
| **HTTP** | Hypertext Transfer Protocol |
| **HTTPS** | Hypertext Transfer Protocol Secure |
| **JSON** | JavaScript Object Notation |
| **API** | Application Programming Interface |
| **IPv4** | Internet Protocol version 4 |
| **IPv6** | Internet Protocol version 6 |
| **DDoS** | Distributed Denial of Service |

| | |
|---|---|
| **FPGA** | Field-Programmable Gate Array |
| **BMv2** | behavioral Model version 2 |
| **INT** | In-band Network Telemetry |
| **IoT** | Internet of Things |
| **NMAE** | Normalized Mean Absolute Error |
| **NFV** | Network Function Virtualization |
| **P4** | Programming Protocol-independent Packet Processors |
| **ONOS** | Open Network Operating System |
| **PLC** | Programmable Logic Controller |
| **SDN** | Software-Defined Networking |

# CONTENTS

# 1 INTRODUCTION

Computer networks are a fundamental part of our lives. Communication issues in large-scale applications, such as social networks, cloud environments, and streaming platforms, can lead to billions of dollars in losses, making every second of instability a critical concern for businesses and users alike (HUANG et al., 2023).

In today's interconnected world, where businesses rely heavily on online services, and consumers expect seamless experiences, network reliability and performance are paramount. Downtime, slow loading times, or service disruptions can have significant financial implications, damage a company's reputation, and result in a loss of customer trust (FRANCO; GRANVILLE; STILLER, 2023). Ensuring the robustness and efficiency of computer networks is a constant challenge for network administrators and engineers (LAHIRI, 2023). They must continually monitor and optimize network infrastructure to maintain high availability, low latency, and fast data transfer rates.

The origin of those instabilities may vary, and despite advancements in technology and network analyzing systems, systems are still vulnerable to disruptions and anomalies (ASSEN et al., 2022). Unfortunately, detecting these issues often happens too late, and the costs of Quality of Service (QoS) damage remain unacceptably high.

If we make a parallel of computer networks with the human body, diseases (*e.g.*, anomalies) have different pathogens, causes, and have different body responses. A disease may start as a fever and end up as an infection that kills or permanently damages the body. It is important to identify diseases in early stages to make quick and effective treatments, preventing their evolution and major damages. In the same fashion, computer networks are vulnerable to many disruptions, such as malicious activities promoted by external agents, malfunctions in the network components, or a bug. Those disruptions may lead to several losses and a decline in QoS, compromising the user experience. To avoid such scenarios, we must detect any deviation in network functionality that might be correlated to a disruption, preventing events that might affect the quality of the system directly. For that, we must fully understand, for example, what kind of behavior a Distributed Denial-of-Service (DDoS) attacks may cause in the switch pipeline, if the attack interferes with ports, headers, timestamps, or any other information that can be analyzed from the device using monitoring techniques. To do so, we rely on Programmable Networks and Software-defined Networking (SDN) (NUNES et al., 2014) technologies, which have evolved to operate computer networks dynamically and enables to collect data of network devices and network flow in a robust and tailored way.

One of the most relevant contributions of the last decade in computer networking is the SDN concept. SDN has revolutionized the way networks are managed and configured, separating the control plane from the data plane (KREUTZ et al., 2015). This approach allows for greater flexibility, scalability, and programmability in network infrastructure. By centralizing network management through software, SDN has enabled more efficient resource utilization, faster network provisioning, and easier implementation of network policies. As a result, it has significantly enhanced network performance and security, making it a pivotal advancement in the field of computer networking.

A relevant aspect of this technology is the wide range of techniques that extract data from the network, especially from networking devices. Back then, forwarding devices were treated as black boxes, and information availability was a subject of suppliers' discretion. Thanks to SDN technology, network management systems can get data from the network at the kernel-level in real-time applications. Open-source communities, such as the P4 Language (BOSSHART et al., 2014), developed frameworks to collect and collate this data. Despite the fact that we are able to collect a large amount of data, there is still need to develop solutions to filter and qualify useful information about networks and devices behaviors.

Different techniques can be used and have been used in the literature to address problems on programmable networks from different perspectives (ZHENG et al., 2023) . One of promising techniques that are not well explored in the context of programmable networks, is the idea of Behavioral Fingerprinting.

Behavioral Fingerprinting is a technique used in network monitoring and cybersecurity to characterize and identify the unique behavior patterns of network devices or entities (SAUERESSIG et al., 2023). Much like how a human fingerprint uniquely identifies an individual based on their distinct patterns, Behavioral Fingerprinting analyzes and captures the behavior of devices, such as switches, routers, or even users, by monitoring their interactions and activities over time. This approach involves collecting and analyzing a wide range of data, including network traffic, communication patterns, latency measurements, queue utilization, and more. By creating a Behavior Fingerprint for each device or entity, network administrators and security professionals gain valuable insights into normal operating behaviors, allowing them to detect anomalies, identify potential threats, and optimize network performance. This proactive approach to network monitoring and security enables quicker identification and response to deviations from established behavioral norms, enhancing the overall robustness and reliability of the network infrastructure (SáNCHEZ HUERTAS CELDRáN, 2021).

In the context of computer networks, Behavior Fingerprint refers to the unique patterns and characteristics exhibited by network devices , such as switches or routers, in their normal operation. Each device has its own specific behavior and response to different network conditions and traffic. By analyzing and understanding these behavioral patterns, network administrators can identify deviations and anomalies that might indicate potential security threats or malfunctions. The Behavior Fingerprint of a network device might include information about how it handles traffic, processes packets, uses computational resources, and responds to various commands and requests. This data is collected through monitoring techniques and can be used to establish a baseline of normal behavior for the device. When anomalies occur (*e.g.*, unusual traffic patterns, unexpected responses, or deviations from the established baseline) (FRANCO et al., 2021) it may indicate the presence of malicious activity, network attacks, or hardware/software issues. Therefore, by continuously monitoring and comparing the device's current behavior to its behavioral fingerprint, network administrators can detect and respond to potential threats or network performance problems proactively.

However, there is still room for approaches that explore Behavioral Fingerprinting to identify and mitigate anomalies. Behavioral Fingerprinting holds the promise of enhancing network monitoring and security by providing a deeper understanding of device behavior beyond traditional monitoring metrics. By capturing and analyzing unique behavioral patterns, such as traffic flow characteristics, protocol usage, and performance metrics, it can offer a more comprehensive and context-aware perspective of network devices' activities.

This Bachelor Thesis proposes *FEVER*, a framework that allows the detection of different types of anomaly by analyzing each switch individually in a network system. *FEVER* establish metrics to be collected and analyze it during an a unknown activity, being able to identify if a switch is "sick" or not. The name *FEVER* comes from the idea that human fever is a detection system able to identify many types of infections from different agents and morphologies(*e.g.*, viruses, bacteria, fungi and even cancer cells). A fever does not tell what kind of anomalies are occurring in the human body, but it is able to tell that something is wrong. While its inspiration is also able to mitigate an anomaly, *FEVER* only detects anomalies. The mitigation process is decided by the network managers, but we do not discard to implement mitigation techniques in future work.

For our test scenario, we used P4 Programs from the P4 Language Tutorials repository available on Github (SAUERESSIG, 2024). The testbed consists of a Mininet virtual network with three switches and three hosts. We have modified the python script of the tutorials responsible by setting up the Mininet environment and the behavioral Model version 2 (bmv2) switches to create a normal flow of packages and elephant flow. We also modified the P4 Programs in order to identify these modifications. To collect resource data about the switches, we have written a shell script that runs the *perf* and *proc/stat/* commands simultaneously, each second it would save resource metrics to csv for each switch during a time of one hour. Mininet threats each switch as a single process, so we identify the Process Identifier (PID) corresponding to the switch activity and then monitors it using *proc/stat/* and *perf*. Then, after one hour running the exercise, we have gather the analyzed data to csv file with the PID of the switch.

The rest of this document is organized as it follows. Section II provides background on theoretical information. Section III discusses related work on programmable networks and Behavioral Fingerprinting. Next, Section IV introduces the *FEVER* approach and provides implementation details, while Section V presents the evaluation of the work. Finally, Section VI concludes the work and provides opportunities for future work.

# 2 BACKGROUND

In this section, we focus on describing concepts and technologies that are relevant to the fully understanding of such a work. This includes concepts related to anomaly detection, computer networks and Behavioral Fingerprinting.

## 2.1 Anomaly Detection and Machine Learning

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior (SANCHEZ et al., 2021). An anomaly, is an event or pattern that deviates significantly from the expected or normal behavior. In other words, it represents an unusual or abnormal activity that differs from the established baseline of typical behavior. The detection of anomalies provides the possibility of taking action against potential misbehavior and security threats, ensuring the integrity and reliability of the network, while also ensure the QoS and Quality of Experience (QoE) for users.

Anomalies can manifest in various ways, including unusual spikes in network traffic, unauthorized access attempts, unexpected data patterns, or irregular system behavior. (SANCHEZ et al., 2021) These deviations from the norm can be indicative of cyberattacks, system malfunctions, or operational errors.

Many techniques are available to detect anomalies. Most of them are Machine Learning (ML) algorithms that identifies deviations and outliers, from simple data statistics to hyperdimensional data (CHANDOLA; BANERJEE; KUMAR, 2009). We have implemented two ML algorithms in order to detect anomalies due their qualified characteristics to the scenario: The One-Class Support Vector Machine (OCSVM) and Local Outlier Factor (LOF).

OCSVM was introduced by (SCHÖLKOPF et al., 1999). A Support Vector Machine can create a non-linear decision boundary to separate two classes. It separates them in the higher dimensional space, projecting data points that cannot be separated by a straight line in a higher dimensional space, so there is a hyperplane that separates the data points of one class from another. OCSVM, on the other hand, has only one class, because it is a unsupervised learning technique and not a usual classifier algorithm. It separates all the data points from the origin in a higher dimensional space and maximizes the distance from this hyperplane to the origin. In other words, the origin is the class that the algorithm tries to separate from the normal class. It adapts to datasets where the normal class significantly outnumbers the anomalous class, by focusing on learning the characteristics of the majority class. It is effective in high-dimensional spaces, making suitable for datasets with a large number of features. *FEVER* works with large amounts of features to be as versatile as possible, so OCSVM is really useful in this context. Also, it has the capability of providing an outlier score for each data point, indicating its distance from the decision boundary, giving a degree of abnormality. This is useful for understanding anomalies in different levels and for helping to understand the early stages of an attack or a malfunction.

(BREUNIG et al., 2000) introduces LOF as an algorithm of anomaly detection. It is local in that the degree depends on how isolated the object is with respect to the surrounding neighborhood. It evaluates the local density of data points relative to their neighbors, identifying outliers based on their deviation from the local context. It measures the extent to which a data point's local neighborhood differs in density compared to its neighbors, making it effective for detecting anomalies in datasets with varying density patterns. Computer Network are highly contextual, having unique traits and behaviors. A virtual network might not present the exact same numbers a real-world network presents, but LOF respect this characteristic due its capability because it considers the local neighborhood of each data point, allowing it to identify outliers relative to their local context. It also performs pretty well when applied to high-dimensional data.

## 2.2 Behavioral Fingerprinting

A research field within behavior data science is focused on creating device behavior patterns (fingerprints) able to optimize their performance and detect potential abnormalities in the early stages. (SANCHEZ et al., 2021) We call this field Behavioral Fingerprinting. It is a collection of metrics of an object with expected values over time. The Behavior Fingerprint of a network device might include information about how it handles traffic, processes packets, uses computational resources, and responds to various commands and requests.

This data is collected through monitoring techniques and can be used to establish a baseline of normal behavior for the device and network applications running. There are two possible application scenarios for it: one consists identifying devices with different granularity levels to differentiate them and fully exploit their capabilities while the second focuses on detecting cyberattacks, malfunction, or misbehavior to mitigate them. In this work, we focus in the second scenario, but *FEVER* can be applied for the first one as well, since it is able to identify different switches. Deciding the scenario is crucial to select a behavior source, metrics that describes a functioning device. They can be externally collected, such as electromagnetic signals, clock drift in time and packet payload; or they can be in-device behavior source, such as hardware events, resource usage and the behavior of softwares and processes inside the device. We use in-device behavior source, as we see in the next sections. All this data needs to be processed, we can use different approaches to do so. The existing techniques are categorized in the following five groups: rule-based, statistical, knowledge-based, Machine Learning (ML) and Deep Learning (DL), and time series approaches. Our approach use ML techniques to process our data. After the data processing, we achieved a detection system that uses the normal behavior of the device as a parameter, the Behavior Fingerprint.

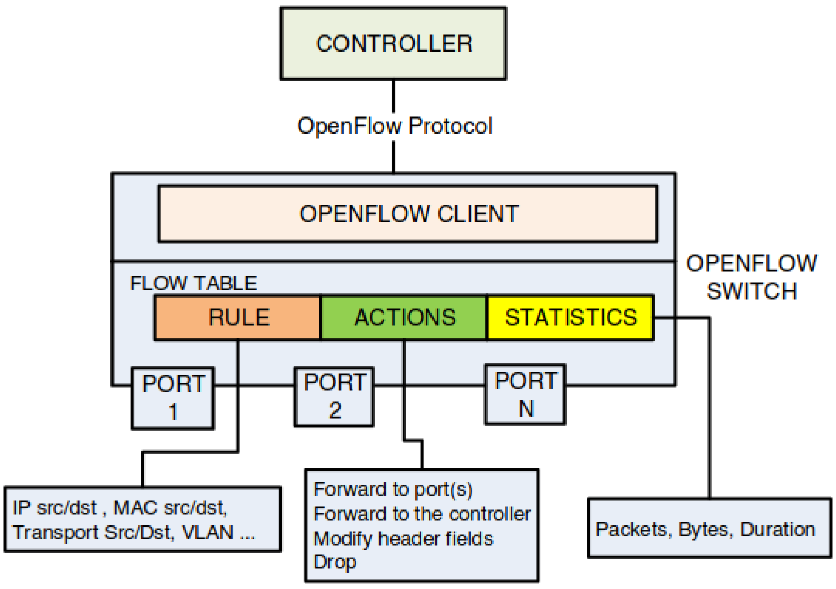
## 2.3 SDN and Programmable Networks

SDN is a revolutionary approach to network management that decouples the hardware forwarding system from control decisions and allows network administrators to control and manage network resources dynamically through software applications. (NUNES et al., 2014) Unlike traditional network architectures, SDN separates the control plane from the data plane, enabling more flexibility, scalability, and programmability. The control plane decides how data is managed, routed, and processed, while the data plane is responsible for the actual moving of data.

One of the most relevant architectures of SDN is OpenFlow. (NUNES et al., 2014) It is an architecture which P4 Programs are based on. Figure 2.1 show how it works. The forwading device called OpenFlow switch has a number of flow tables (*e.g.*, rule, actions and statistics) and it is connected to the controller by the OpenFlow Protocol, a standardized communication protocol used in SDN to facilitate the exchange of information between the control plane and the data plane of network devices. Flow tables defines how the packets belonging to a flow will be processed and forwarded. The flow tables have match rules for incoming packets, counters to collect statistics and a set of instructions, or actions, to be executed when a match is done.

Upon a packet arrival at an OpenFlow switch, packet header fields are extracted and matched against the matching fields portion of the flow table entries. If a matching entry is found, the switch applies the appropriate set of instructions, or actions, associated with the matched flow entry. If the flow table look-up procedure does not result on a match, the action taken by the switch will depend on the instructions defined by the table-miss flow entry.

Figure 2.1: Overview of an Openflow Switch



Source: (NUNES et al., 2014)

This cutting-edge technology has the potential to significantly enhance Behavior Fingerprinting and anomaly detection in network environments. SDN's programmability allows it to trigger automated responses when anomalies are detected. For example, it can isolate affected devices, reroute traffic, or alert security personnel, reducing response time and minimizing potential damage. Also, SDN's ability to gather and process large amounts of network data makes it an ideal environment for machine learning-based anomaly detection. By leveraging machine learning algorithms, SDN can continuously learn from network behavior and detect previously unseen threats effectively.

## 2.4 Mininet

Mininet was introduced by (LANTZ; HELLER; MCKEOWN, 2010). It is a tool for rapidly prototyping networks, visioning a flexible, deployable, scalable and realistic SDN prototypes. Mininet creates a virtual network by placing host processes in network namespaces and connecting them with virtual Ethernet pairs. A virtual Ethernet pair, or veth pair, acts like a wire connecting two virtual interfaces; packets sent through one interface are delivered to the other, and each interface appears as a fully functional Ethernet port to all system and application software. Veth pairs may be attached to virtual switches such as the Linux bridge or a software OpenFlow switch. A host in Mininet is simply a shell process ( *e.g.*, bash ) moved into its own network namespace with the unshare(CLONE NEWNET) system call. Each host has its own virtual Ethernet interface(s) (created and installed with ip link add/set) and a pipe to a parent Mininet process (*e.g.*, mn) which sends commands and monitors output. Software OpenFlow switches provide the same packet delivery semantics that would be provided by a hardware switch. Both user-space and kernel-space switches are available.

Mininet can be used to emulate a network system to test our anomaly detection system. Prototype behavior should represent real behavior with a high degree of confidence; for example, applications and protocol stacks should be usable without modification, according to (LANTZ; HELLER; MCKEOWN, 2010). Thanks to this feature, Mininet is able to reconstruct a reliable testbed to analyze a system in Behavioral Fingerprinting process.

## 2.5 Programming Protocol-independent Packet Processors (P4)

P4 is a high-level language for protocol-independent packet processors based on the OpenFlow architecture. (BOSSHART et al., 2014) It was designed to reach three main goals: Reconfigurability, the controller ability of changing the parsing and processing of packets in switches; Protocol independence, so the switch can be untied to any specific network protocol; and finally target independence, the ability to describe packet processing functionality independently of the specifics of the underlying hardware. The typical P4 program pipeline consists of stages like parsing, matching, action execution, and deparser. In the parsing stage, incoming packets are dissected into fields, and the matching stage identifies relevant patterns based on header information. Actions specify the processing to be applied, such as modifying headers or forwarding packets. The deparser stage formats the processed packet for transmission. P4 has an Application Programming Interface (API) called P4Runtime, which is a control plane specification for controlling the data plane elements of a device defined or described by a P4 program. It is used together with a P4 program to configure a programmable switch.

## 2.6 Bmv2 Switch

Behavioral model version 2 (bmv2) is a virtual switch used to operates its forwarding process according to a P4 Program. According to (FOUNDATION, 2023), this is the second version of the reference P4 software switch and it is written in C++11. It takes as input a JavaScript Object Notation (JSON) file generated from a P4 program by a P4 compiler and interprets it to implement the packet-processing behavior specified by that P4 program. It runs on a general purpose Central Processing Unit (CPU) such as Intel/AMD/etc.

It implements the P4 features listed in the table 2.1. They correspond to the P4_14 language specification, and also with P4_16 plus the v1model architecture, which is intended to match the architecture defined in the P4_14 language specification.

The specific target used in this work was simple_switch_grpc, which can accept TCP connections from a controller, where the format of control messages on this connection are defined by the P4Runtime API specification and it is used for debug purposes. In a Linux environment, a simple_switch_grpc process corresponds to a switch parsing packages according to a P4 compiled program. This process can be used to gather the performance metrics of these switches for building a Behavior Fingerprint.

Table 2.1: P4 Features Implemented on BMv2

| **Feature** | **Description** |
|---|---|
| Counters | Tracks the number of packets or bytes that match specified conditions. |
| Meters | Measures and regulates the rate of packet flow based on configured rules. |
| Registers | Storage for maintaining state information accessible by P4 programs. |
| Action Profiles | Grouping of actions that can be executed together, enhancing code modularity. |
| Action Selectors | Mechanism for selecting one action from a set of actions to execute. |
| Hash and Checksum Functions | Various functions for computing hash values and checksums. |
| Pseudo-Random Number Generation | Generates random numbers for specific use cases in P4 programs. |
| Digest Messages | Messages generated by the data plane and sent to the control plane for communication. |
| Switch Architecture | Components include ingress parser, ingress control, packet buffer with packet replication engine, egress control, and egress deparser. |
| Multicast Replication | Replicates packets for multicast transmission to multiple destinations. |
| Cloning/Mirroring of Packets | Creates copies of packets for analysis or monitoring purposes. |
| Resubmit Packets | Redirects packets from the end of ingress back to the start of ingress. |
| Recirculate Packets | Redirects packets from the end of egress back to the start of ingress. |

# 3 RELATED WORK

In this section, we discuss different work with focus on building device fingerprint and also data collectors that have been proposed to assist network managers to increase efficiency and network protection. We also provide an analysis of approaches for anomaly detection and network security management that are related to this work.

## 3.1 Behavioral Fingerprinting

In (BAI; KIM; REXFORD, 2022), the authors introduce P40f, a passive OS fingerprinting tool that runs directly on programmable switch hardware to identify the Operating Systems (OS) running on hosts in a network. OS fingerprinting is helpful for managing enterprise networks, detecting vulnerabilities, and applying security policies based on the OS type. Passive fingerprinting is preferred over active methods, as it monitors network traffic in real-time without introducing additional network load. P40f is implemented using the P4 language on Intel Tofino switch hardware. It uses TCP header and option fields in TCP SYN packets to perform OS fingerprinting, similar to the software-based tool p0f. Unlike traditional software-based passive fingerprinting tools, P40f can process traffic at high line rates and take direct actions, such as dropping, rate-limiting, or redirecting traffic, based on the identified OS, without needing control-plane messages. However, P40f is very specialized and specified to detect OS-specific vulnerabilities and does not gather information about non-host devices.

In another work, (SáNCHEZ HUERTAS CELDRáN, 2021) uses Raspberry Pi devices as a testbed for anomaly detection and device recognition. The main goal of this work is to verify how feasible is building a solution to identify the sensors belonging to a crowdsensing platform in and to check how does the solution scale when the number of devices deployed increases. It has also investigated the most suitable data sources of resource-constraint devices and ML/DL algorithms to create Behavior Fingerprints detecting anomalies produced by threats and if it is possible to build a common ML/DL-based system that uses device Behavior Fingerprinting to detect anomalies produced by heterogeneous cyberattacks affecting different resource-constrained devices of crowdsensing platforms.

Other approaches also consider bmv2 switches as tested to implement approaches for fingerprinting. In (KUZNIAR; NEVES; HAQUE, 2022), the authors proposed FingerP4, a solution to identify events from seven different Internet of Things (IoT) devices entirely in the data plane. Next, PoirIoT (KUZNIAR et al., 2022) was proposed as a more robust solution for IoT device detection based only on packet metadata (*e.g.*, length and direction) and could detect several devices as soon as it exchanges its first packets in the network. The work was also implemented in a Tofino-based programmable switch.

While there is existing work that focuses on fingerprinting, only some of it concentrates on analyzing devices from an individual device perspective. Based on our investigations, the identification of anomalies at an earlier stage requires monitoring each device individually to prevent anomalies from propagating throughout the network. Therefore, there are need for developing of techniques that enable the network management system to identify if a forwarding device is behaving abnormally.

## 3.2 Collection of Metrics

Metrics are an important part of Behavioral Fingerprinting. In this section we analyze works that collect metrics for diverse uses. An effective way to collect information about devices and architectures in programmable networks is to use In-band Network Telemetry (INT). Due to its fine-grained monitoring, INT can generate a high report rate, leading to many report packets sent to the collector.

The INTCollector is introduced in (TU et al., 2018) to address this issue and efficiently process INT telemetry reports. It stores INT metric values in a time-series database using InfluxDB. This database is selected for its high write throughput, support for custom timestamps, and push mechanism, allowing efficient data storage. The key components of the implementation are the event detection mechanism, which detects significant changes in network metrics, and the exporter, which sends metric values to the database periodically or when new events occur. INTCollector can help with this to some extent. Collecting and analyzing the INT metadata over time can build a profile of the switch's behavior. For example, it can track the switch's handling of flows, latency experienced at different hops and queue occupancy patterns. However, INTCollector focuses mainly on the INT Framework and does not propose hardware monitoring.

For resource monitoring, (NGO et al., 2003) proposes a novel monitoring tool called Wireless Ad-hoc Network Monitoring Tool (WANMon), which allows the user to monitor the resource consumption by ad-hoc wireless network based activities. It can be installed in a wireless node to monitor many metrics, such as the number of packets and bytes that are sent and received by the wireless interface, and protocol used; the power consumption in sending and receiving data either for the purposes of the networking applications running on the node or for routing the data of the other network node; how much memory, over a past period of time, was used to send and receive data and CPU usage in kernel space for networking applications.

In another work, (DONG et al., 2019) focused in CPU performance to build a graph based device fingerprint scheme for devices identification and authentication. The experiment consisted in analyzing 10 identical PCs. Each PC has two CPUs, 2 GB Random Access Memory (RAM) and Hard Disk Drive (HDD). Considering the effect of data access speed of the local RAM and reading/writing speeds of the HDD, the paper proves that each CPU has a singular graph, identifying different devices.

All of these works provide data collectors or metrics that can be used to build device Behavior Fingerprints since more data available can help approaches to identify anomalous behaviors in order to identify technical problems in automated ways. It is also possible to use different approaches concomitantly, adapting them for a better use by specific frameworks.

## 3.3 Anomaly Detection

The following section aims to show different approaches of implementations to detect anomalies in computer networks. Many techniques have been implemented, using information of different metrics. We present some works that each one have a different approach for anomaly detection.

With the increasing connectivity of Programmable Logic Controllers (PLCs) in the industrial IoT, cybersecurity threats have received attention. (HAN et al., 2021) presents a method to detect and defend against cyberattacks targeting PLCs, specifically focusing on denial of service (DoS) attacks and control-logic injection attacks. The proposed approach monitors CPU usage to identify abnormal temporal behavior and employs a control-flow analysis to detect stack-based buffer overflow attacks. Implemented in a water tank control system, experimental results demonstrate the effectiveness of the method in enhancing system security with minimal overhead.

(GULENKO et al., 2016) proposes a system architecture for real-time anomaly detection in large-scale Network Function Virtualization (NFV) systems built upon the OpenStack platform, a widely used open-source cloud software, to enhance monitoring and analysis for real-time anomaly detection. The basic data to collect represents the usage of different resources in each host. This includes CPU and RAM usage, I/O operations of different partitions and mount points, and network I/O metrics of different network interfaces and protocols. The analysis involves supervised and non-supervised classification methods, with preliminary experiments showing a high percentage of correctly identified anomaly situations. The proposed framework, integrated into OpenStack, activates pre-defined countermeasures upon anomaly detection to mask or repair outages or degraded performance.

We observed that there is no a *de facto* approach for anomaly detection. Most of the works specialize in a type of anomaly or they depend of specific metrics. In real world, anomalies appear in different forms and sometimes at the same time. It is important to propose approaches to address such a gap by proposing architectures and methodologies able to handle different anomalies using a diverse set of metrics.

## 3.4 P4-based Cybersecurity Applications

We conducted also a systematic literature, particularly of applications for detecting and mitigating cyberattacks and managing cybersecurity in programmable networks. We focused on three main aspects: *(i)* network availability, *(ii)* network security, and *(iii)* privacy.

P4 programs have been developed to ensure greater network availability and perfect functioning from a performance and security point of view. Such applications include protecting against impersonation attacks by filtering malicious traffic (LI et al., 2019), identifying DDoS attacks by statistically analyzing traffic flows (DING; SAVI; SIRACUSA, 2021), and also verifying P4 programs using static checks (DUMITRESCU et al., 2020) and assertions (WANG et al., 2023) to identify possible execution faults. However, most of this work focuses primarily on traffic analysis only or has too specific use cases for cyberattack detection. Such solutions also have limitations in identifying dynamic changes on the network, such as when a P4 program is maliciously replaced or changed by network operators or even when a potential cyberattack is imminent (*e.g.*, anomalous behaviors started). Thus, we argue that identifying anomalies at an earlier stage requires an intelligent monitoring of devices individually to prevent anomalies from propagating throughout the network.

Therefore, although there are several emerging applications for P4 programs and programmable networks, there is still a need for automated solutions that allow for proper monitoring and management of the security of existing networks and services. Current challenges include limitations in memory usage and accessibility for P4 program development and using collected metrics for effective performance and security applications (HAUSER et al., 2023). There is also the need to make the network more robust and autonomous, which involves combining telemetry and ML elements to create infrastructures that adapt to the needs of the network and can predict possible failures. Such elements can support better detection of cyberattacks and anomalies while improving the detection performance of interested behaviors.

ML can be an ally to address such issues due to its potential to understand complex data patterns and adapt to heterogeneous scenarios based on different training datasets (USAMA et al., 2019). (ALMEIDA; PASQUINI; VERDI, 2021) presents an experiment that combines Machine Learning (ML) algorithms with INT to estimate service metrics in computer networks. The main goal is to improve the management of networks and services by utilizing the fine-grained data provided by INT in combination with ML techniques. Specifically, the experiment focuses on estimating the QoS of a video streaming service called DASH (Dynamic Adaptive Streaming over Hypertext Transfer Protocol (HTTP) ), which is known for its ability to adapt video quality according to network conditions.

The experiment was conducted using a virtualized environment with network components and virtual machines. Data packets were sent with INT metadata appended at each hop, allowing for detailed network metrics collection. Three load patterns were used: sinusoid, flashcrowd, and a mix of both. ML models, including Decision Tree (DT), Random Forest (RF), K-nearest neighbors (KNN), and Neural Networks (NN), were applied to estimate the QoS metrics based on the INT data. The results showed that the Random Forest (RF) algorithm outperformed other ML models, achieving a Normalized Mean Absolute Error (NMAE) of below 10 % for estimating the QoS of the video streaming service. Notably, the buffer-related metadata had the most influence on the learning model, and the network node closest to the client had the most accurate information for predicting video metrics.

Overall, the experiment demonstrated the potential of using ML together to improve the estimation of service metrics in computer networks. The combination of ML algorithms, INT data, and resources usage can enhance network management and service quality. The opportunity has therefore arisen to implement approaches based on ML to analyze statistical metrics and resource usage behavior in order to identify anomalies before such behavior becomes a problem for the operation of services in programmable networks (SAUERESSIG M. F. FRANCO, 2023), such as a DDoS attack or a malicious change to a network program. To do this, we can use the concept of Behavioral Fingerprinting, which, although there are applications of the concept in other scenarios (*e.g.*, malware detection in IoT scenarios) (SANCHEZ et al., 2021), is still is underinvestigated in the context of programmable network security.

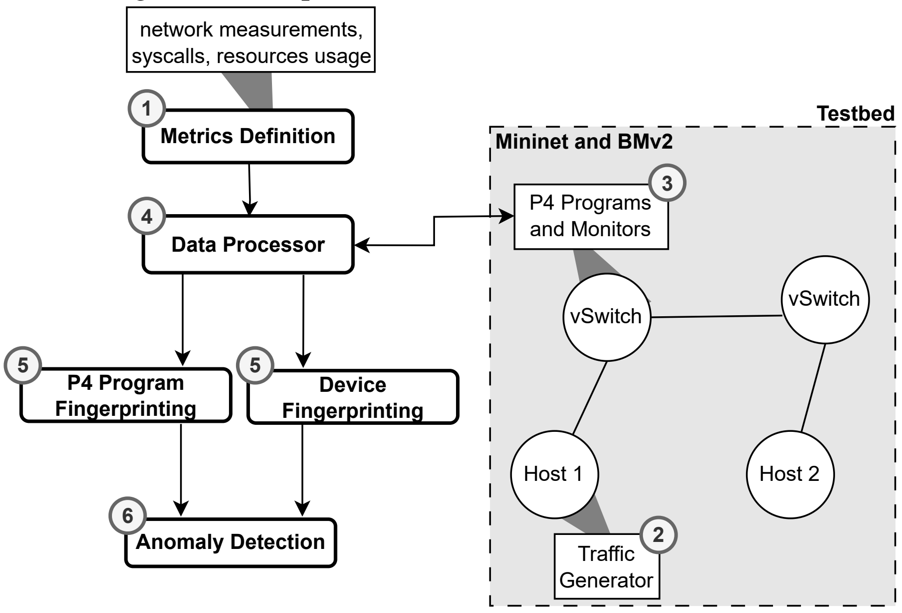
# 4 APPROACH

This section details the approach of our methodology and its mechanisms. *FEVER* proposes and implements an approach to explore programmable device Behavioral Fingerprinting for anomaly detection. When anomalies occur (*e.g.*, unusual traffic patterns, unexpected responses, or deviations from the established baseline), it may indicate the presence of malicious activity, network attacks, or hardware/software issues. Therefore, by continuously monitoring and comparing the device's current behavior to its Behavior Fingerprint, *FEVER* users can detect and respond to potential threats or to network performance problems proactively.

## 4.1 *FEVER* Methodology

In Figure 4.1 we can see the workflow of *FEVER*. It consists of different steps of analysis, from creating a testbed to processing the data collect on it. *FEVER* works in such way you can test many types of environment and collect different types of data that might help us to identify problems in the device, leading to a diagnosis of the network whose devices are presenting anomalous behavior.

Figure 4.1: Proposed Workflow for *FEVER* framework



Source: Original results from the research

First, we must define metrics related to the devices of our network. They can be syscalls, resource usage metrics that can be obtained with *perf*, *proc* or *top* in a virtual network running in a Linux system or specific commands of the device that give us how much memory and CPU is being used by the switch.

After choosing the metrics we want to monitor and collect, we must create our testbed. We choose a topology and generate traffic that simulates a normal network behavior. It is also viable to use a real network as testbed. Then we monitor the activity of the switches in the tesbed using monitoring programs or P4 Programs in the case of using INT. We monitor the network for sufficient time, which can be many hours, and then we process the data collected from the testbed in a organized way. We then select relevant features and discard the ones which has high correlation. The Behavior Fingerprint is ready, and we use it to train ML models to identify anomalies in the network. These models can run in real time to detect whenever an anomaly appears in a device from the network.

Each of the steps proposed by the *FEVER* methodology is discussed in details in the rest of this chapter, including implementation details and examples of usage scenarios.

## 4.2 Metrics Definition

This step of *FEVER* consists in selecting metrics that might be relevant for the analysis. These metrics can be from different sources and have different natures. A network manager must ask the question: "In the presence of this type of anomaly, what kind of characteristics may exhibit anomalous behavior?". For example, a flood attack might create longer queues or use more instructions to process each package flooding the device. A malware spreading through the network might allocate memory in the switch system. Depending on the nature of the anomaly, the network manager may choose some types of metrics or, if they want to identify the maximum number of different anomalies, they might use all of the metrics available, however metrics collection might be an expensive task to complete.

The metrics must be related to switch since we are predicting the devices behavior individually. They must tell something about the device behavior. Resource usage metrics such as memory allocated by the switch to do its operations tells us how much memory the switch is currently using in its network environment.

An example of metrics definition would be choosing to monitor the memory and CPU used by the switch, the queue length and the switch ID. The queue length is useful to see how much traffic the switch is receiving, the CPU and memory tells us about how much resources the switch is allocating to process the packages and the Switch ID helps to distinguish the multiple switches in the network. Table 4.1 is an example table of metrics and their purposes on the behavior analysis

Table 4.1: Overview of Initial Metrics Considered for Behavioral Fingerprinting

| **Metric** | **Purpose of Usage** | **Monitoring Method** |
|---|---|---|
| CPU | The CPU usage is related to an abnormal increase or decrease of instructions in case of an anomaly. Bugs and malicious attacks can increase the usage of CPU. | Linux top command |
| RAM | RAM monitoring helps us to detect anomalies if they allocate memory to do malicious activity or due to misconfiguration. | Linux top command |

After the set of metrics is defined, the network manager must find a way to collect the metrics they will be monitoring. This can be achieved using software or scripts specialized in collecting data from switches. Now the network manager must generate traffic to trace a normal behavior of the network devices.

## 4.3 Traffic Generator

Traffic generation is used to set a normal setup of the network so we can understand the behavior of the devices in a daily basis and distinguish it from an anomaly. This can be achieved simulating a network in a virtualized environment (*e.g.*, Mininet), creating a real-life network that behaves similarly to the real one, or using a functional architecture and its daily traffic to monitor it. It is also possible to emulate an traffic with anomalous activity, such as flood attacks or elephant flows in order to discard irrelevant features or to choose better ML models that makes more precise predictions given the anomalous scenario.

In an emulated traffic, we must establish relations between the switches and hosts (*e.g.*, who sends package to who) and the bandwidth of the connections. This is useful for emulating attacks, since flood attacks for example can increase the bandwidth in order to preclude the network functionality, so we need a baseline to distinguish the normal activity from an anomaly. In a real-life scenario, it would be useful to know the bandwidth of the network if attacks simulations would be conducted.

There many ways to generate traffic to monitor our testbed (*e.g.*, HTTP requests/responses, File Transfer Protocol (FTP) or Secure Copy Protocol (SCP) and Domain Name System (DNS) Requests). One useful way is using iperf/iperf3, a tool for active measurements of the maximum achievable bandwidth on IP networks. It supports tuning of various parameters related to timing, buffers and protocols (*e.g.*, Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Stream Control Transmission Protocol (SCTP) with Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6) ). For each test it reports the bandwidth, loss, and other parameters. (DUGAN ELLIOTT, )

We can create UDP clients and servers using iperf and then send packages to different hosts. Using a python script, we can automate this feature and emulate different hosts communicating and switch forwarding packages like in real world.

## 4.4 Monitoring and Data Processing

While the traffic flows, the network manager must collect the metrics from the switch. To do so, we use monitoring tools, such as specialized softwares or scripts that execute commands that informs about the switch behavior. One can use Linux commands such as *proc* and *perf*, which give us information about the resource usage of processes, and collect the resource metrics with shell scripts. We can use Scapy, a Python library for manipulating packets to sniff packages and extract headers information, which are inserted in package headers.

Here is an example of bash script that access how many threads a switch process has and its virtual size memory:

```
PROCESS_STAT=($(sed -E 's/\([^)]+\)/X/' "/proc/$PID/stat"))
PROCESS_THREADS=${PROCESS_STAT[20]}
PROCESS_VSIZE=${PROCESS_STAT[23]}
```

This consists of using Regular Expressions to isolate the values and save them as an array. This array is organized in such way that their elements correspond to the values that we want (*e.g.*, Process Virtual Size in the 23 position of the array). As you can see, the data is extracted and organized in a efficient way, so can be stored later on.

For more precise data collection, some hours might be necessary to monitor the network. Networks with high complexity and with different behavior along the day may need much more monitoring. There are some scenarios where a network along the week shows specific bandwidth which should not be considered as an anomaly, but simply a usual pattern for this time span.

It is also the case to consider different time spans of data collection. Monitoring each second gives us a more precise and granular analysis of the device, however if we monitor each 5 seconds, we have a better notion of the average values of each metric.

The data processing step consist in saving the data in such organized way so we can understand it. Naturally, the data also must be organized so it can be used for training the ML models for anomaly detection. The monitor collects the data and saves it into a csv file for example, the file name must be referring to the switch or the process related to the switch.

## 4.5 Anomaly Detection and Behavioral Fingerprinting

After processing and selecting the most relevant data, we achieved a Behavior Fingerprint, which means we know what is a normal behavior for the given devices. This means we can use this fingerprint to analyzes and detect outliers in the device behavior.

In this step, we use the Behavior Fingerprint to train an unsupervised ML model to detect anomalies. There are many options of models, it is the manager task to choose the most viable model to detect anomalies using the features selected by they. It is important to use Data Analysis tools such as plots and correlation matrices to see what kind of model would benefit with the behavior analyzed. For example, Local Outlier Factor was used in our evaluation scenario because we have identified isolated regions with anomalous activity while looking the Resident Set Size Memory graphic. This algorithm performs really well with outliers in isolated points. We also used the One-Class Support Vector Machine because it copes great with high-dimensional data, and we used several features to analyze our behavior. After the tuning step, the anomaly detection is ready to be deployed on a real life network. We must monitor and gather data while the network flows and we must put it on test using our anomaly detection system.

In the next section, we evaluate *FEVER* using a Proof-of-Concept (PoC) implementation to show how it handled the detection of anomalous traffic flow and P4 program alterations. For that, different scenarios are built and evaluated in a quantitative and qualitative way.

# 5 EVALUATION

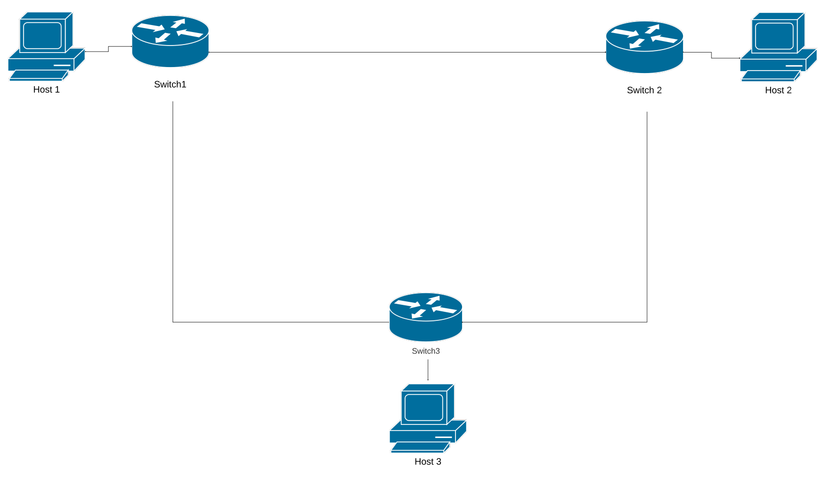
In this section we present all evaluations conducted in order to test our approach regarding its performance in identify anomalous behavior in both switch processing and P4 programs. All experiments were conducted in a virtualized scenario, using Mininet and BMv2 switches, as described below.

## 5.1 Initial Setup

We conduct a set of experiments to prove the feasibility of the proposed approach for Behavioral Fingerprinting of programmable networks. For that, we built our testbed using on well-known resources and applications for programmable networks. The setup consists a Mininet virtual network with three bmv2 switches and three hosts, running applications from the P4 Language tutorials repository (FOUNDATION, 2023), most precisely the Multi-hop Route Inspection (MRI) implementation. The topology considered is shown in Figure 5.1. A Python script was developed to setting up the Mininet environment and configure the hosts and bmv2 switches. Also, *iperf* (SIATERLIS; GARCIA; GENGE, 2012) was integrated to the script in order to allow the creation of different flow behaviors according to the tests needs, such as a normal flow of packets between all switches and specific elephant flows.

To collect resource data from the switches, we have written a shell script that runs the Linux *perf* and *proc* commands simultaneously. The script collects metrics each second and save the collected data to .csv file for each switch during a time of one hour. For that, the PID of each virtual switch is identified. As Mininet treats each switch as a single process, it is possible to identify the PID corresponding to the switch activity and then monitored it using *proc* and *perf* commands. Thus, after one hour running the simulation, we have gathered the analyzed data to csv file with the PID of the switches. The normal behavior of the network is established according to the configuration available in Table 5.1. Therefore, each experiment consisted of running the p4lang Makefile tutorial with modifications to generate the desired traffic (*cf.* Table 5.1) for 1 one hour. The behaviors considered involve *(i)* normal traffic, *(ii)* anomalous traffic, and *(iii)* modified traffic. Table 5.1 shows how the normal traffic is defined in terms of source/destination and size. The *(ii)* anomalous traffic is defined as a elephant flow that floods the network using the maximum available throughput. For the *(iii)* modified behavior, we consider a

Figure 5.1: Proposed Topology for Experiments



Source: Original results from the research

MRI with modifications, such a conditional branch and arithmetic operations.

Table 5.1: Traffic and Connections Considered for a Scenario of a Normal Traffic Flow

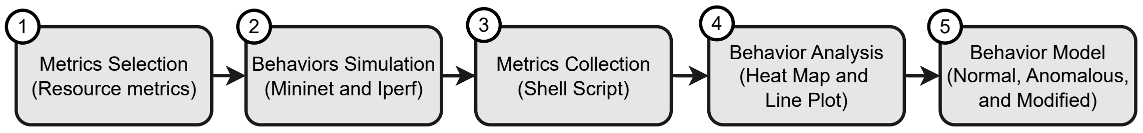
| **Source Host** | **Destination Host** | **MB/s** |
|---|---|---|
| Host 1 | Host 2 | 1 |
| Host 1 | Host 3 | 10 |
| Host 2 | Host 3 | 10 |
| Host 2 | Host 1 | 10 |

Source: Original results from the research

The experiments have to be run a number of times until the behaviors can be precisely modeled. For each round, we create a heat-map of all metrics and drop the metrics with high correlation. Next, a line bar is plotted and a manual analysis is performed in order to classify and calibrate the model behavior. For our experiments, we ran three times for each behavior in order to create our training dataset for normal, anomalous, and modified behaviors. Therefore, the experiments were ran nine times in total, with 1 hour duration each. An overview of the methodology applied is depicted in Figure 5.2.

Step 1 comprises the selection of metrics to be considered (*e.g.*, RAM memory and CPU usage). Next, in Step 2, the Mininet environment is deployed and the network traffic generated according to the behavior demands. A shell script monitors, as Step 3, the defined metrics in real-time during 1 hour and save it for further analysis. For the analysis, in Step 4, different plots are performed in order to identify high correlated

Figure 5.2: Overview of the Methodology to Create Fingerprinting of Normal, Anomalous, and Modified behavior.


1
Metrics Selection (Resource metrics)
2
Behaviors Simulation (Mininet and Iperf)
3
Metrics Collection (Shell Script)
4
Behavior Analysis (Heat Map and Line Plot)
5
Behavior Model (Normal, Anomalous, and Modified)

Source: Original results from the research

metrics and understand the current behavior. At this step, some metrics can be removed and other can be added for a next round of experiments. Finally, in Step 5, the behavior model is defined, which comprises metrics that are relevant to highlight changes in the behavior based on the traffic (normal and anomalous traffic behavior scenarios) and also P4 programs running in the network (*e.g.*, modified scenario).

## 5.2 Feature Selection

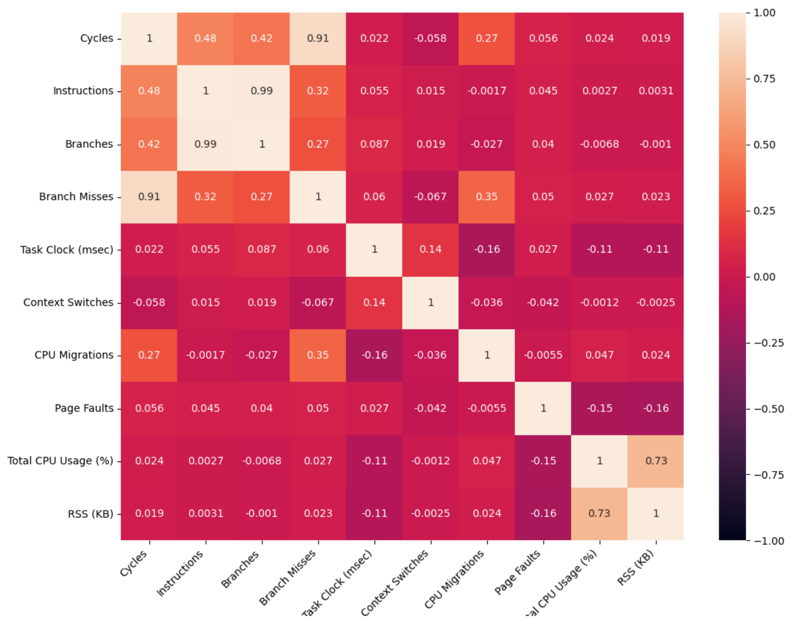
After the selection of metrics and monitoring activities (Steps 1–3), the feature selection have to be performed. Feature selection is an important step for establishing a Behavior Fingerprint of the device. It consists of gathering the data defined by the metrics selections and choosing the most relevant ones using tools to analyze correlation between the features. After doing the feature selection, the Behavior Fingerprint is ready for training a ML model for predicting anomalies.

The metrics were gathered using *perf* and *proc* commands in Linux. The *perf* metrics collected consist on information with high granularity in the hope of detecting small fluctuations in the behavior of the switch. Those include time elapsed during the *perf* command execution, cycles, instructions, branches, branch misses, task clock, context switches, CPU migrations and page faults. The *proc* metrics used were the process virtual size, total memory usage, Resident Set Size (RSS) in KB and the percentage of RSS used.

We have also used a heatmap to show the correlations between metrics, in order to drop features that were not relevant when preparing the dataset to train the model. Heatmaps are useful for feature selection because we can identify high correlated features that may impact in the quality of the predictions. High correlated features might provide redundant information, resulting in overfitting. Therefore, it is important to remove such kind of features to achieve better model performance. The darker the color in heatmap is, the more unrelated the feature is compared to other ones. Thus, features with colors closer to white (1) should be removed.

It is important to mention that was verified that some features reduced their correlation with traffic intensity. For example, instructions have shown high correlation to another feature in a low package traffic environment, while this correlation is reduced when we intensified the traffic. Thus, we did not drop features that shown a reduction of correlation in certain environments.

Figure 5.3: Features Correlation Heatmap



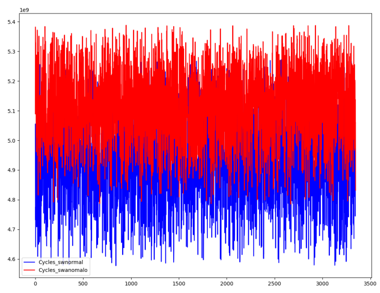
Source: Original results from the research

Based on that, the Time Elapsed (s), RSS (in percentage), Virtual Memory Size, Total Memory Usage (in percentage) where dropped due their high correlation with the other features. The resulting correlation heatmap is shown in Figure 5.3. For example, it can be observed that there is only one light diagonal in the matrix with the value 1, showing that the high correlation is only between the feature itself. The features with 99 % correlation were maintain since their not correlated to other values more than one, and due the high granularity of the values.

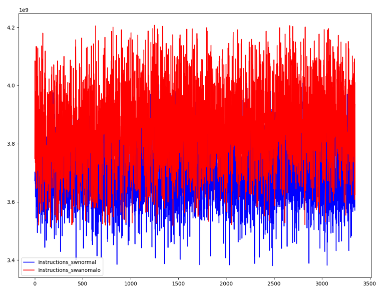
After the heatmap analysis, a line plot is provided in order to do a more in-depth analysis. Some features might present overlapping or unidentifiable anomalous behavior, which will need to be dropped to improve the performance of some ML algorithms.

The most perceptive difference in visual analysis occurred with the anomalous behavior against normal behavior. However, most of these features were overlapping, which can present a challenge for ML algorithms to recognize an anomaly. The overlapping is

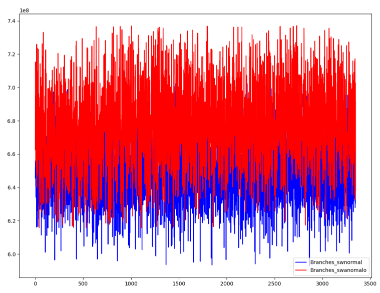
Figure 5.4: Analysis of Overlapping Features



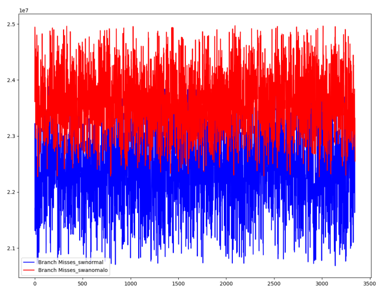
(a) CPU Cycles



(b) CPU Instructions



(c) Branches



(d) Branch Misses

Source: Original results from the research

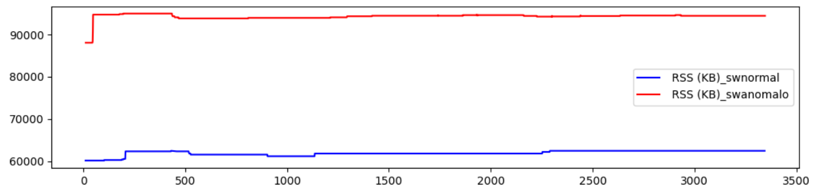
shown in Figure 5.4. The most relevant feature was the RSS in KB, in both anomalous traffic and modified traffic, whose lines never touched the normal traffic one, as depicted in Figure 5.5 (a) and (b).

Therefore, based on the feature selection it is possible to determine from a set of metrics, which metrics can be useful for the process of Behavioral Fingerprinting. Although we used this set of metrics for our experiments, additional metrics can be also considered and be analyzed following the process as explained above. Evaluations conducted in different applications scenarios are provided in the next sections. All of the scenarios considers the feature selection approach provided in this section.

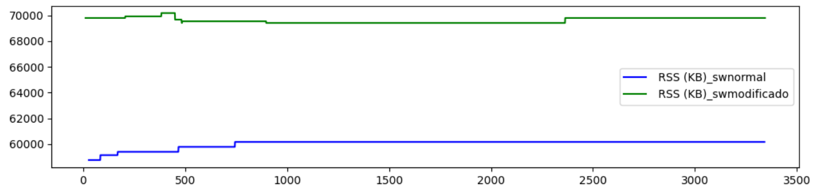
Finally, we use the selected features to build the Behavior Fingerprint and train the model for anomaly detection, including anomalous traffic and modified P4 programs. In the rest of this section, we will introduce scenarios considered and provide information about the evaluations performed for these different scenarios.

For training our models, we have considered two ML algorithms: OCSVM and LOF. These algorithms ran using the remaining features after the feature selection. Some features were dropped temporarily to check how do the models performance behave with the absence of useful features. 10 % of the data was used as the testing set, while the

Figure 5.5: RSS Comparison between Switches in Different Scenarios



(a) Switch receiving Anomalous Traffic vs. Switch with Normal Traffic



(b) Switch running Modified P4 Program vs. Switch Running a Regular P4 Program

remaining 90 % was used as the training set. The data was shuffled.

## 5.3 Detection of Anomaly Traffic

In this section we analyze the performance of our anomaly detection using the anomalous traffic switch behavioral fingerprint. Therefore, the goal is to detect abnormal increases in the data flow through a switch in order to identify possible flooding attacks earlier, such as Ping Flood , SYN/ACK Flood and HTTP or Hypertext Transfer Protocol Secure (HTTPS) Flood.

To create an anomalous traffic, we have used the command *iperf* to send UDP packets. The default behavior of *iperf* when using UDP is to send data as fast as possible, without any specific rate limiting, *e.g.*, it floods the network with UDP packets to measure the maximum throughput. For the context of this experiment, we will call this anomalous behavior as "elephant flow". We have considered this scenario ideal since it would not disrupt the network, but will send enough traffic to be detected. In a real world scenario, it would be beneficial to detect an increase of flow before the DDoS caused by the data flood.

Both host 1 and host 2 behaved like clients and host 3 the server. Both host 1 and host 2 sent *iperf* UDP packets to host 3, creating a high traffic flow through switch 3, the one connected to host 3. Since the elephant flow was directed to host 3, we analyzed only its switch to detect the anomaly, since neither host 1 nor host 2 had enough traffic flow.

To calculate the performance of our models (also used for the other scenarios), we have used the precision, recall, and F1-score metrics. Such a metrics calculation are defined as follows:

$$\text{True Positive Rate} = \frac{\text{Number of correctly identified anomalies}}{\text{Total number of anomalies}}$$

$$\text{False Positive Rate} = \frac{\text{Number of normal instances incorrectly identified as anomalies}}{\text{Total number of normal instances}}$$

$$\text{Precision} = \frac{\text{Number of correctly identified anomalies}}{\text{Total number of instances identified as anomalies}}$$

$$F1 = \frac{2 \times \text{Precision} \times \text{True Positive Rate}}{\text{Precision} + \text{True Positive Rate}}$$

The results using OCSVM shown an accuracy of 100 $\%$. Since it is a ML algorithm designed for situations where you only have examples of the normal class during training, it learns a boundary around the normal instances and classifies anything outside this boundary as an anomaly. The magnitude difference between the RSS memory helped the algorithm to detect perfectly the anomaly. The lack of overlap in the RSS memory feature could indicate a distinct pattern for the anomalous process in this particular aspect. OCSVM may have learned a boundary that effectively captures this separation. This can be verified if we drop the RSS feature. In this scenario, OCSVM performances poorly, with a recall of 0.43 and a false-positive rate of 0.56.

The LOF also scored the same as OCSVM. LOF, in particular, is designed to identify local deviations from the majority of data points. If anomalies form distinct clusters or have noticeable local differences and low density, LOF can excel in detecting them.

We conclude that memory analyze is a reliable source to detect flow changes, preventing flood attacks. With more packages, more memory needs to be allocated in order to operate the switch process properly.

## 5.4 Detection of Changes on P4 Programs

We have implemented slightly different versions of P4 codes to see how much *FEVER* could detect a modified version of the original code.

This scenario was tested to all switches, since the code running on them is the same. The idea is to identify modifications in the behavior of the switch if the code is changed. In a real world scenario, a malicious agent might have access to the P4 programs and might be able to change it to execute malicious activities or incapacitate the program.

In the first modification we have introduced few conditional branches and multiplications, to see how these operations would affect instructions and memory. We will call this modification 1.

```
        // Introduce variable calculations
        bit<32> someVariable;
        someVariable = 100 * 2323;

        // Use someVariable in conditional branches
        if (someVariable > 100) {
            // Perform additional actions or calculations
            hdr.swtraces[0].qdepth = hdr.swtraces[0].qdepth * 1313;
        } else {
            // Perform other actions or calculations
            hdr.ipv4.ihl = hdr.ipv4.ihl * 4;
        }

    }
```

This code instantiate a variable with a value 100 and it is multiplied in a equation. This value is then multiplied by 2323. After that the code passes through a conditional branch where the value instantiate as *qdepth* is multiplied by 1313. *qdepth* is a value that represents the package queue depth length in the switch. Otherwise the Internet Header Length(IHL) value would be multiplied by 4. This values have little impact in the CPU and memory usage, but since we were dealing with features with high granularity and precision, it is important to see how small changes impacts the network and if we can detect it properly. We call this modification 1.

The second modification performed involves instantiating three registers, two of 32 bits and one of 64 bits. This represents a more substantial change since it allocates useless registers, two 32 bits and one 64 bits. Even that P4 already have mechanisms to inform if there are useless registers in the code, this scenario helps to see memory fluctuations in the switch in case of changes. We call this modification 2.

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    register<bit<32>>(2048) reg1;
    register<bit<32>>(2048) reg2;
    register<bit<64>>(50000) reg3;
    ...
```

Unfortunately, the modification 1 (*e.g.*, multiplication scenario) was too feeble to be detected by our model. The changes in instructions, cycles and other features were too similar to the normal switch. In our tests scenarios, we had 1 false positive in switch 3, which was observed in RSS memory.

In the modification 2, we have prominent results for every single switch. As shown in Figure 5.5, the registers allocation was noticed by the RSS memory analysis, in which the OCSVM and LOF models had similar results with this dataset. LOF achieved a 100% accuracy score in switch 1 and switch 2 and a recall of 0.88 in switch 3. OCSVM has scored a little worse in switch 3, with 0.67 of recall, however it scored perfectly in switch 1 and 2.

Memory has shown again to be a reliable source of behavior information about switches. Alterations in code means more or less variables and registers allocated, altering the memory allocation, thus changing the behavior of the RSS in the switch.

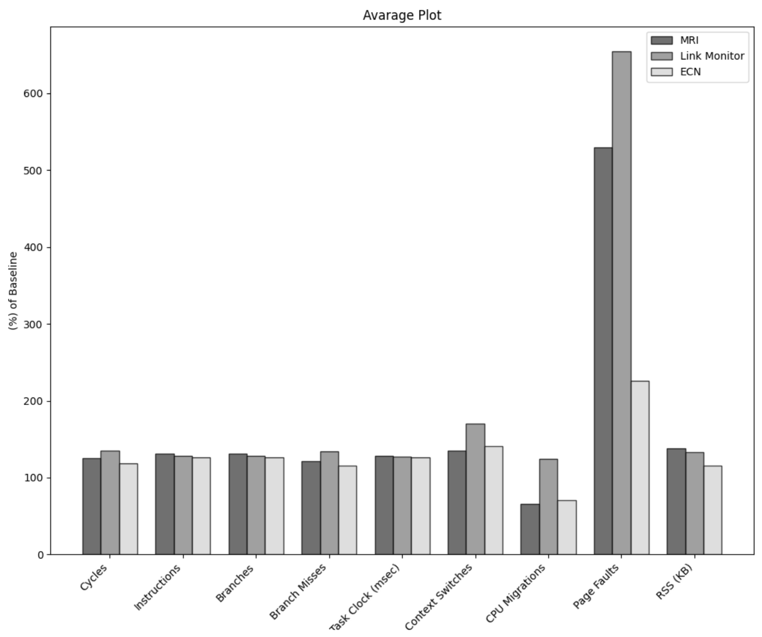
The training dataset used for our evaluation scenarios is available at our git repository (SAUERESSIG, 2024). We made it so everyone interested can replicate our results and implement their own experiments with the *FEVER* methodology.

## 5.5 Detection of P4 Programs

We have conducted other experiments to see the fingerprint of individual P4 programs. We have tested with many examples of the P4 Tutorials, such as a link monitor and an Explicit Congestion Notification (ECN) application. Each program has a individual set of instructions and memory behavior, which enabled us to identify each program individually using ML models.

Figure 5.6 shows an comparison of the features related to a baseline. It consists in the average quantity of each single feature related to a baseline value. The baseline

Figure 5.6: P4 program features comparison to baseline program features



Source: Original results from the research

experiment is a basic forwarding exercise from P4 Tutorials. The idea is to compare the simplest implementation of a P4 program with the more elaborated ones. The value is a percentage of how bigger or smaller are the feature values compared to baseline values.

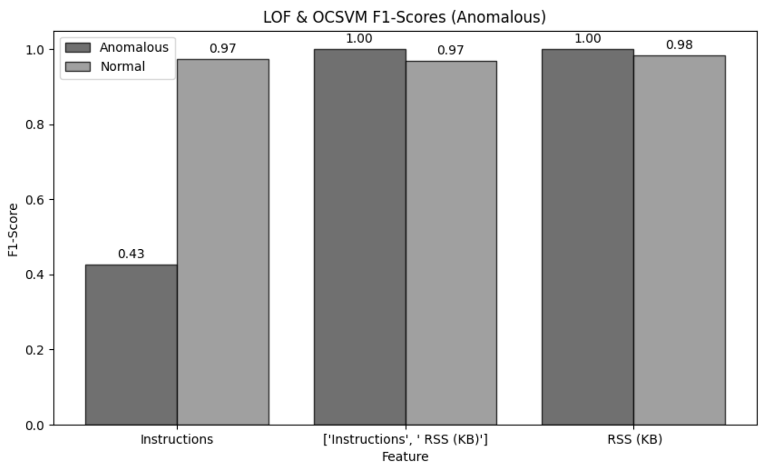
Using a switch running the baseline code as a normal behavior, we have identified MRI, ECN and Link Monitor as anomalies using LOF and OCSVM with perfect F1-Score. In this scenario, dropping the RSS feature compromised so little of the F1-Score, dropping to around 0.95. This shows us that other features are in fact relevant in analyzing P4 programs and trying to create their Behavior Fingerprint.

## 5.6 Discussion and Limitations

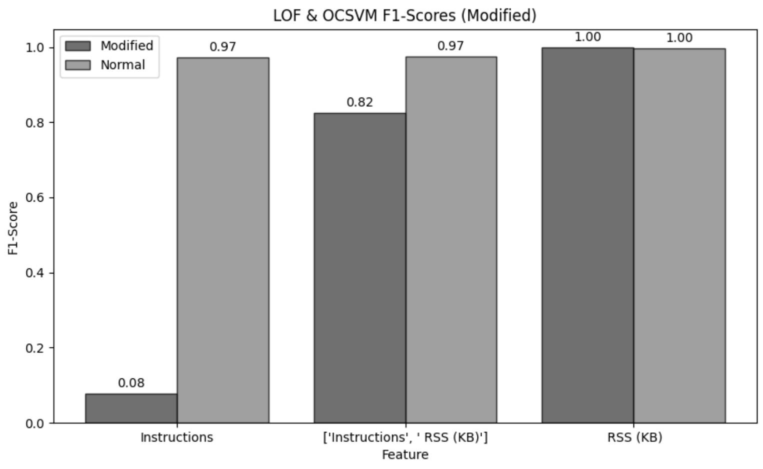
After gathering many metrics to analyze the behavior of the switch, the RSS memory in KB has shown to be the most relevant feature, proving that analyzing memory usage of switches might be a great source of information in order to identify if there is a abnormal traffic flowing through the switch or if the P4 program running in the switch has been modified in some manners. Figure 5.7 shows how relevant RSS was to the F1-Score compared to other features in anomaly detection. Standing alone, it has surpassed detection relying on other features, even itself combined to other features.

The importance of memory may be related to buffering the package received by the switch. With more intense traffic flowing through the switch, more packages must be buffered. The RSS present in the modified version of MRI was increased due the fact we only allocated more memory in the code. However, relying only on features high relevance by dropping other features might overfitting the model since the nature of an anomaly is unknown. Future work may investigate why RSS memory change so much in those scenarios and what other types of anomaly this feature can help to detect and how to avoid overfitting.

Figure 5.7: Analysis of F1-Scores using Features



(a) LOF and OCSVM F1-Scores (Anomalous)



(b) LOF and OCSVM F1-Scores (Modified)

Source: Original results from the research

It is also important to mention that we have identified the abnormality in one single device, which might different from approaches that consider links or many devices to identify a abnormal traffic. *FEVER* has achieved high granularity in traffic analysis and it has show versatility, being able to identify code alterations as well.

The identifiability degree of P4 code changes can still be improved, since this work is exploratory in nature, which means we were able to identify changes when allocating many registers, but we were unable to identify what is the minimum change necessary to detect a modified code.

As a limitation, it is essential to note that our evaluations are conducted in the emulated environment; therefore, due to different abstractions and technical aspects, there are challenges to implementing it in real-world devices (*e.g.*, Tofino and FPGA). For example, our emulated environment uses one single PID for a switch, while many processes are involved in a switch functionality in real life. However, adaptations are possible - from metrics collection to Behavioral Fingerprinting - to efficiently apply our framework in various scenarios and types of devices. The research papers resulted as part of this work are also highlighted in Appendix A.

# 6 CONCLUSION AND FINAL REMARKS

This Bachelor Thesis proposed *FEVER*, a framework for the Behavioral Fingerprinting of programmable networks, including detecting the misbehavior of P4-based switches and P4 programs. ML-based models were employed together with statistical processing to map and understand metrics that highlight potential anomalies given a traffic and a set of P4 programs running. This allows the identification of *(i)* anomalous traffic, *(ii)* malicious changes in the P4 program's code, and the *(iii)* replacement of P4 programs to disrupt the network functionality and associated services. In real-world scenarios, all of these anomalies can happen in parallel, thus making clear the need for automated ML models ready to infer from different data patterns. Like the fever in the human body, our approach highlights abnormal activity in the system that might compromised it by using ML.

As limitation, there still the need to implement a real-world scenario (*e.g.*, Tofino and FABRIC testbed) with real-time detection. For now, *FEVER* only detected anomalies from data collected for hours and do not response during the network functioning. Also, it is based on emulation using Mininet and bmv2. Therefore, although the approach can fit other scenarios, the metrics collected are still dependent on such technologies. Others experiments with more complex networks and interactions must be analyzed, as well other types of anomalies and ML models can be explored for a more accurate prediction.

In conclusion, our experiments provide essential insights into the fingerprinting of programmable networks, and our *FEVER* framework has proven to be a suitable methodology for analyzing the behavior of programmable switches and P4 programs, which can be adapted to real-world scenarios. We look forward to anomaly detection systems improving and mitigation techniques development, so in the future we have not only detection (*e.g.*, *FEVER*) but also an entire immunologic system for computer networks that allows for mitigation of imminent threats

Future work includes: *(i)* investigation of the sensitivity of detection of small changes on P4 program codes and *(ii)* analysis of additional metrics for Behavioral Fingerprinting, including full integration of INT framework and syscalls to provide more information to represent complex behaviors better, *(iii)* real time anomaly detection, *(iv)* using *FEVER* for anomaly mitigation. Furthermore, implementation on real-world scenarios composed by Tofino switches is envisioned.

## REFERENCES

ALMEIDA, L. C. de; PASQUINI, R.; VERDI, F. L. Using Machine Learning and In-band Network Telemetry for Service Metrics Estimation. In: **IEEE 10th International Conference on Cloud Networking (CloudNet 2021)**. Cookeville, TN, USA: IEEE, 2021. p. 33–39.

ASSEN, J. V. D. et al. CoReTM: An Approach Enabling Cross-Functional Collaborative Threat Modeling. In: **IEEE International Conference on Cyber Security and Resilience (CSR 2022)**. Virtually: IEEE, 2022. p. 189–196.

BAI, S.; KIM, H.; REXFORD, J. Passive OS Fingerprinting on Commodity Switches. In: **IEEE 8th International Conference on Network Softwarization (NetSoft 2022)**. Milan, Italy: IEEE, 2022. p. 264–268.

BOSSHART, P. et al. P4: programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, jul 2014.

BREUNIG, M. M. et al. LOF: identifying density-based local outliers. **SIGMOD Rec.**, Association for Computing Machinery, New York, NY, USA, v. 29, n. 2, p. 93–104, may 2000.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 41, n. 3, jul 2009. ISSN 0360-0300.

DING, D.; SAVI, M.; SIRACUSA, D. Tracking Normalized Network Traffic Entropy to Detect DDoS Attacks in P4. **IEEE Transactions on Dependable and Secure Computing**, IEEE, v. 19, n. 6, p. 4019–4031, 2021.

DONG, S. et al. CPG-FS: A CPU Performance Graph Based Device Fingerprint Scheme for Devices Identification and Authentication. In: **IEEE CyberScience and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech 2019)**. [S.l.]: IEEE, 2019. p. 266–270.

DUGAN ELLIOTT, A. M. P. P. **iPerf - The ultimate speed test tool for TCP, UDP and SCTP**. <https://iperf.fr/>.

DUMITRESCU, D. et al. bf4: Towards Bug-Free P4 Programs. In: **SIGCOMM 2020**. Virtually, USA: [s.n.], 2020. p. 571–585.

FOUNDATION, O. N. **P4Language Repository**. 2023. <https://github.com/p4lang>.

FRANCO, M. et al. SecGrid: A Visual System for the Analysis and ML-Based Classification of Cyberattack Traffic. In: **IEEE 46th Conference on Local Computer Networks (LCN 2021)**. Edmonton, Canada,: IEEE, 2021. p. 1–8.

FRANCO, M. F.; GRANVILLE, L. Z.; STILLER, B. CyberTEA: a Technical and Economic Approach for Cybersecurity Planning and Investment. In: **36th IEEE/IFIP Network Operations and Management Symposium (NOMS 2023)**. Miami, USA: IEEE/IFIP, 2023. p. 1–6.

GULENKO, A. et al. A System Architecture for Real-time Anomaly Detection in Large-scale NFV Systems. **Procedia Computer Science**, v. 94, p. 491–496, 2016. 11th International Conference on Future Networks and Communications (FNC 2016) / The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016) / Affiliated Workshops.

HAN, S. et al. Anomaly Detection Based on Temporal Behavior Monitoring in Programmable Logic Controllers. **Electronics**, v. 10, n. 10, 2021.

HAUSER, F. et al. A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research. **Journal of Network and Computer Applications**, v. 212, p. 103561, 2023.

HUANG, K. et al. **The Devastating Business Impacts of a Cyber Breach**. 2023. Harvard Business Review, <https://hbr.org/2023/05/the-devastating-business-impacts-of-a-cyber-breach>.

KREUTZ, D. et al. Software-Defined Networking: A Comprehensive Survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14–76, 2015.

KUZNIAR, C. et al. IoT Device Fingerprinting on Commodity Switches. In: **IEEE/IFIP Network Operations and Management Symposium (NOMS 2022)**. Budapest, Hungary: [s.n.], 2022. p. 1–9.

KUZNIAR, C.; NEVES, M.; HAQUE, I. IoT Device Fingerprinting on Commodity Switches. In: **Dalhousie Computer Science In-House Conference**. Budapest, Hungary: IFIP, 2022. p. 1–9. Poster Session.

LAHIRI, K. **Why Cybersecurity Should Still Be A Top Priority For Businesses**. 2023. Forbes, <https://tinyurl.com/forbesBusiness2023>.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In: **9th ACM SIGCOMM Workshop on Hot Topics in Networks**. Monterey, California: [s.n.], 2010. p. 1–6.

LI, G. et al. NETHCF: Enabling Line-Rate and Adaptive Spoofed IP Traffic Filtering. In: **IEEE 27th International Conference on Network Protocols (ICNP 2019)**. Chicago, USA: IEEE, 2019. p. 1–12.

NGO, D. et al. WANMon: a resource usage monitoring tool for ad hoc wireless networks. In: **28th Annual IEEE International Conference on Local Computer Networks, 2003. LCN '03. Proceedings.** Bonn/Konigswinter, Germany: IEEE, 2003. p. 738–745.

NUNES, B. A. A. et al. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. **IEEE Communications Surveys Tutorials**, v. 16, n. 3, p. 1617–1634, 2014.

SANCHEZ, P. M. et al. A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets. **IEEE Communications Surveys Tutorials**, v. 23, n. 2, p. 1048–1077, 2021.

SAUERESSIG, M. et al. An Approach for Behavioral Fingerprinting of P4 Programmable Switches. In: **Anais da XX Escola Regional de Redes de Computadores**. Porto Alegre, RS, Brasil: SBC, 2023. p. 55–60.

SAUERESSIG, M. F. F. M. **FEVER-P4 Repository**. 2024. <https://github.com/ComputerNetworks-UFRGS/FEVER-P4>.

SAUERESSIG M. F. FRANCO, E. J. S. L. Z. G. M. An Approach for Behavioral Fingerprinting of P4 Programmable Switches. In: **XX Escola Regional de Redes de Computadores (ERRC 2023)**. Porto Alegre, Brazil: [s.n.], 2023. p. 22–60.

SCHÖLKOPF, B. et al. Support Vector Method for Novelty Detection. **Advances in neural information processing systems**, v. 12, 1999.

SIATERLIS, C.; GARCIA, A. P.; GENGE, B. On the use of Emulab testbeds for scientifically rigorous experiments. **IEEE Communications Surveys & Tutorials**, IEEE, v. 15, n. 2, p. 929–942, 2012.

SáNCHEZ HUERTAS CELDRáN, B. M. P. S. S. Secure Crowdsensing Platforms Through Device Behavior Fingerprinting. Ediciones de la Universidad de Castilla-La Mancha, p. 87–90, 2021.

TU, N. V. et al. INTCollector: A High-performance Collector for In-band Network Telemetry. In: **14th International Conference on Network and Service Management (CNSM 2018)**. Rome, Italy: IEEE/IFIP, 2018. p. 10–18.

USAMA, M. et al. Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges. **IEEE Access**, IEEE, v. 7, p. 65579–65615, 2019.

WANG, Q. et al. Foundational Verification of Stateful P4 Packet Processing. In: SCHLOSS-DAGSTUHL-LEIBNIZ ZENTRUM FÜR INFORMATIK. **14th International Conference on Interactive Theorem Proving (ITP 2023)**. [S.l.], 2023.

ZHENG, C. et al. In-Network Machine Learning Using Programmable Network Devices: A Survey. **IEEE Communications Surveys  Tutorials**, p. 1–35, 2023.

# APPENDIX A — RESEARCH PAPERS

Two research papers were developed from collaborations in the context of this Bachelor Thesis. The research papers were submitted and accepted for regional and international conferences in order to collect feedback from the community. This allows for the validation of the methodology and results generated by this Bachelor Thesis, including improvements based on technical reviews and collaborations.

## A.1 Accepted Paper – AINA 2024

M. Saueressig, M. F. Franco, E. J. Scheid, A. Huertas, G. Bovet, B. Stiller, L. Z. Granville: FEVER: Intelligent Behavioral Fingerprinting for Anomaly Detection in P4-based Programmable Networks; International Conference on Advanced Information Networking and Applications (AINA-2024), Kitakyushu, Japan, October 2023, pp. 1-12.

- **Title:** *FEVER: Intelligent Behavioral Fingerprinting for Anomaly Detection in P4-based Programmable Networks*
- **Contribution:** A refined framework for Behavioral Fingerprinting and anomaly detection, including experiments showing the detection, using ML models, of changes in P4 programs and anomalous traffic.
- **Abstract:** The evolving computer network landscape has enabled programmability in various network aspects, including Software-defined Networking (SDN) for control plane programmability and the introduction of the Programming Protocol-independent Packet Processors (P4). P4, a vendor-independent protocol, allows programmability on the data plane, offering flexibility for new services and applications. However, this flexibility introduces the need for automated solutions to monitor and manage the security of evolving networks and services. In this work, we propose FEVER, a framework utilizing P4-based telemetry and network device (switch) resource consumption to create fingerprints of network and P4 application behaviors. FEVER provides a comprehensive approach to identifying network anomalies through various metrics. The framework was evaluated in a virtualized scenario using unsupervised Machine Learning (ML) algorithms to detect diverse P4 program behaviors and traffic overload, demonstrating its potential for early detection of malicious activities in programmable networks. The results indicate high accuracy in identifying misbehavior and detecting sudden changes in P4 programs.

- **Status:** Accepted for publication
- **Qualis:** A3
- **Conference:** 38th International Conference on Advanced Information Networking and Applications (AINA-2024)
- **Date:** April 17 – April 19, 2024
- **Local:** Kitakyushu, Japan
- **URL:** To appear
- **Digital Object Identifier (DOI):** To appear

## A.2 Published Paper – ERRC 2023

M. Saueressig, M. F. Franco, E. J. Scheid, L. Z. Granville: An Approach for Behavioral Fingerprinting of P4 Programmable Switches; XX Escola Regional de Redes de Computadores (ERRC 2023), Porto Alegre, Brazil, October 2023, pp. 55-60.

- **Title:** *An Approach for Behavioral Fingerprinting of P4 Programmable Switches*
- **Contribution:** Discussions on opportunities for Behavioral Fingerprinting on programmable networks and proposal of a framework.
- **Abstract:** Behavioral Fingerprinting is a technique used to understand the behavior of devices, enabling a better understanding of their functionality and improved anomaly detection. This paper proposes a methodology for generating the Behavior Fingerprint of programmable switches. The methodology outlines the process of selecting metrics for analysis, extracting data from them, and organizing the information to construct a Behavior Fingerprint for a programmable device within a network.
- **Status:** Published
- **Qualis:** -
- **Conference:** 20ª Escola Regional de Redes de Computadores (ERRC 2023)
- **Date:** October 23 - October 25, 2023
- **Local:** Porto Alegre, RS, Brasil
- **URL:** <https://sol.sbc.org.br/index.php/errc/article/view/26004>
- **Digital Object Identifier (DOI):** <https://doi.org/10.5753/errc.2023.913>