An Approach for Behavioral Fingerprinting of P4 Programmable Switches

Matheus Saueressig, Muriel F. Franco, Eder J. Scheid, Lisandro Z. Granville

Institute of Informatics (INF) – Federal University of Rio Grande do Sul (UFRGS)
Av. Bento Gonçalves, 9500, Porto Alegre – RS – Brazil

`{matheus.saueressig, mffranco, ejscheid, granville}@inf.ufrgs.br`

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 behavioral 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 behavioral fingerprint for a programmable device within a network.*

1. Introduction

Computer networks are evolving at an unprecedented pace, driven by the relentless demands of our increasingly interconnected world. Programmable networks are emerging as a crucial response to these evolving requirements, offering the flexibility and adaptability needed to support the diverse services and applications that rely on network infrastructure [Nunes et al. 2014]. Prominent examples of programmable networks and enablers are the Software-defined Networking (SDN) and Network Functions Virtualization (NFV) concepts [Bondan et al. 2019].

A relevant aspect of programmable networks is the wide range of techniques that extract data from the network, especially from networking devices [Sánchez et al. 2021]. Back then, forwarding devices were treated as black boxes, and information availability was a subject of suppliers' discretion. Thanks to the SDN technology, network management systems can interact directly with the control plane, while Programming Protocol-independent Packet Processors (P4) switches [Bosshart et al. 2014] allow having a programmable data plane. Open-source communities, such as the P4 Language, provide frameworks to collect and collate data from such a data plane [Tan et al. 2021], thus allowing us to understand network patterns and behaviors fine-grained. Even though we can collect and process a large amount of data, we still need to develop solutions to filter and qualify helpful information to understand networks and device behaviors (*i.e.*, behavioral fingerprinting).

Behavioral fingerprints in the context of programmable networks can refer to the unique patterns and characteristics exhibited by network devices and network applications in their regular operation, such as P4 programmable switches running a P4 program. Each device has its specific behavior and response to different network conditions and traffic. By analyzing and understanding these behavioral patterns, network operators can identify deviations and anomalies that might indicate potential security threats or malfunctions [Sánchez et al. 2021]. However, this is a challenging task since a massive amount of data and metrics must be collected, processed, and correlated for an accurate analysis.

Therefore, there is still room for approaches that explore behavioral fingerprinting to identify and mitigate anomalies. Behavioral fingerprinting promises to enhance 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, behavioral fingerprinting can offer a more comprehensive and context-aware perspective of network activities.

Thus, in this work, we propose an approach for behavioral fingerprinting of programmable switches and P4 applications to identify anomalies in programmable networks by looking at the behaviors of network devices and applications. For that, the approach *(i)* defines a set of metrics (*e.g.*, network-centric and resources consumption) to be used for the behavioral fingerprinting of programmable networks, *(b)* describes a clear path to generate the fingerprints, and *(c)* implements monitors to collect the relevant metrics. A set of traffic behaviors is also defined and generated for the collection process. The experimental testbed was built using Mininet [Lantz et al. 2010], bmv2 [Open Networking Foundation 2023], and P4 applications running on a Commercial Off-The-Shelf (COTS) server. A case study is conducted as a preliminary evaluation to show the feasibility and potential applications of the proposed approach.

The rest of this work is organized as follows: Section 2 presents related work. Our approach is described in Section 3, and finally, the conclusion and future work are presented in Section 4.

2. Related Work

In this section, we discuss examples of two different ways of building a device fingerprint and examples of data collectors that have been proposed to assist network managers to increase efficiency and network protection.

2.1. Fingerprinting

In [Bai et al. 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.

Other approaches also consider bmv2 switches as tested to implement approaches for fingerprinting. In [Kuzniar et al. 2022b], the authors proposed FingerP4, a solution to identify events from 7 different IoT devices entirely in the data plane. Next, PoirIoT [Kuzniar et al. 2022a] 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.

2.2. Collection of Metrics

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, mainly 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.

While there is existing work that focuses on fingerprinting, only some of it concentrates on analyzing devices from an atomized perspective. Identifying anomalies at an earlier stage requires monitoring each device individually to prevent anomalies from propagating throughout the network. Therefore, we must develop techniques that enable the network management system to identify if a forwarding device is behaving abnormally.

3. Approach

This work proposes and implements an approach to explore programmable device behavior fingerprinting for anomaly detection. Behavior fingerprinting can be defined as a collection of metrics of an object with expected values over time. The behavioral 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. 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 behavioral fingerprint, network administrators can detect and respond to potential threats or network performance problems proactively.

Figure 1 shows the framework pipeline, with the different steps highlighted as numbers. Programmable network devices have many metrics available. Therefore, in Step 1, selected metrics must be chosen for analysis. This process involves understanding the different metrics available and the scenarios in which the fingerprinting will be applied. Next, to conduct the collection in a controlled environment, a testbed is built using Mininet and bmv2 [Open Networking Foundation 2023]. In Step 2, a traffic generator is implemented to emulate different traffic patterns for behavior analysis. Therefore, different P4 programs and monitors (Step 3) can interact with traffics from different sources

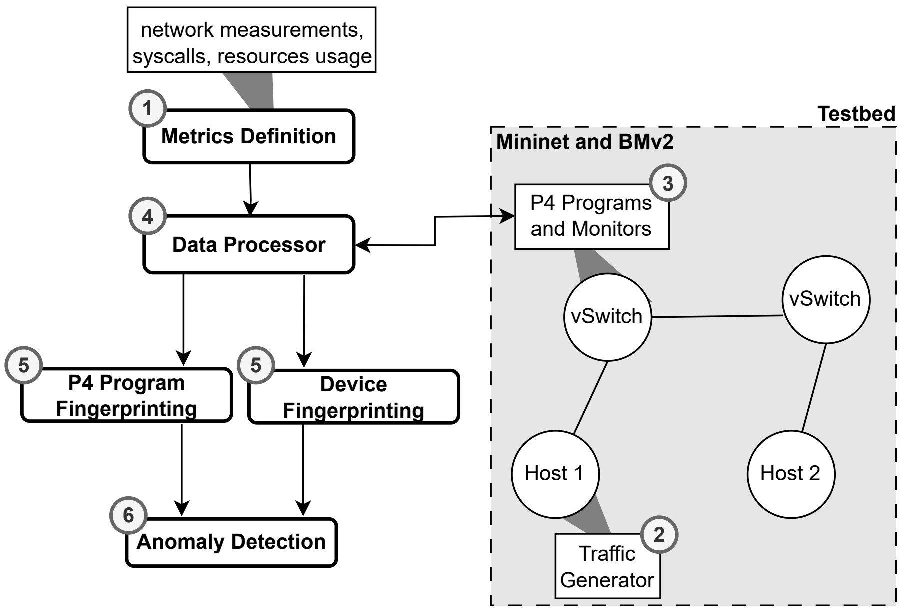

Figure 1. Proposed Approach for Behavioral Fingerprinting

containing specific characteristics. Although the monitoring is focused on device activities, network activity is crucial to observe how the device behaves in certain situations (*e.g.*, a DDoS attack, regular behavior, or running specific P4 programs).

For metrics collection, we use P4 Programs, monitors, or both. P4 Programs can collect device metrics thanks to packet manipulation, allowing us to add more headers with information about how the device processed the packet. Monitors can collect metrics, such as hardware activity and resource consumption. After the data collection, it must be processed so that it is organized to be used as training data for a Machine Learning model. The component called the Data Processor (Step 4) is in charge of such an organization and analysis.

This organized data is then used to generate the behavioral fingerprinting, separated into P4 Program Fingerprint and Device Fingerprint (Step 5). This distinction is suggested to enable bug detection in P4 Programs and to avoid confusing it with other types of anomalies. However, there is no restriction on attempting to identify patterns in both the device and P4 program files together. Finally, this behavioral fingerprint is used to train an ML model to identify an expected network flow and whether this network is suffering from an anomaly. The anomaly detection (Step 6) using the generated fingerprinting is also performed. However, this is out of the scope of this work.

4. Case Study

To validate our framework, we have developed a data pipeline focused on monitoring how anomalies alter the CPU and Memory usage in individual switch devices and compare this situation with a regular data flow. For this experiment, we have made a switch behavior fingerprint, analyzing the CPU and memory usage of processes related to bmv2 while running a Mininet network. We run a topology of three switches and five hosts. Both switches run a P4 program called Multi-Hop Route Inspection (MRI).

MRI enables users to monitor the routes taken by packets and their associated queue lengths. We use the implementation of MRI from the p4lang community [Open Networking Foundation 2023]. This adds an identifier and queue length to the header stack of each packet. When the packet reaches its destination, the sequence of switch identifiers corresponds to the route taken, with each identifier followed by the queue length of the switch's port.

In alignment with that, Process Identifiers (PIDs) corresponding to the bmv2 activity of compiling and forwarding the packets received in the virtual network were identified along this work. While the hosts established a connection and a packet flow, we monitored the CPU and memory usage using the bmv2 PIDs. For precise monitoring, we have isolated the processes to avoid a long-running process consuming much CPU on the system where the bmv2 software switches are running, slowing down the experiment and affecting the performance measurements.

Table 1 summarizes the collected metrics and composes an example of behavior fingerprinting built using our approach. The data collected corresponds to a switch memory consumption, CPU usage, and queue length over time. We have collected one hour of traffic to standardize normal behavior. After that, anomalies (*e.g.*, cyberattacks, misconfigurations, and P4 bugs) will be introduced in the network to observe changes in the switch processing behavior.

Table 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
Queue Depth	Anomalies might affect packet processing by slowing it, thus increasing the congestion. The Package Queue may increase if an anomaly changes the behavior of the switch.	INT
Switch ID	The Switch ID allows us to separate information for each individual switch (*e.g.*, queue, CPU, and RAM).	INT

For the collection of CPU and RAM memory, the Linux Top (Table of Processes) command was used since it allows to select specific process for analysis. This allows for the collection of metrics for the previously identified PIDs (*i.e.*, bmv2 activities). Also, INT is used to collect information though the MRI implementation. These monitoring methods allows to collection all metrics needed for this case study. Additional and more accurate mechanisms and tools can be used for further analysis.

5. Conclusions and Future Work

With the multitude of data types available in modern network architectures, there arises a need for standardized data collection to optimize data interpretation and enhance the efficiency of network monitoring. This paper proposes a methodology that employs behavioral fingerprinting to understand better the workflow of programmable switches in a network context.

In future work, we will generate behavioral fingerprints based on different network flows and P4 programs to validate the overall approach. Also, supervised machine learning techniques will be considered to detect anomalies using the generated fingerprints, thus helping to identify misbehavior in programmable devices, P4 programs, or even imminent attacks in the network.

References

Bai, S., Kim, H., and Rexford, J. (2022). Passive os fingerprinting on commodity switches. In *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, pages 264–268.

Bondan, L., Franco, M. F., Marcuzzo, L., Venancio, G., Santos, R. L., Pfitscher, R. J., Scheid, E. J., Stiller, B., De Turck, F., Duarte, E. P., Schaeffer-Filho, A. E., d. Santos, C. R. P., and Granville, L. Z. (2019). FENDE: Marketplace-Based Distribution, Execution, and Life Cycle Management of VNFs. *IEEE Communications Magazine*, 57(1):13–19.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Kuzniar, C., Neves, M., Gurevich, V., and Haque, I. (2022a). IoT Device Fingerprinting on Commodity Switches. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9.

Kuzniar, C., Neves, M., and Haque, I. (2022b). IoT Device Fingerprinting on Commodity Switches. In *Dalhousie Computer Science In-House Conference*, pages 1–9. Poster Session.

Lantz, B., Heller, B., and McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. In *9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6.

Nunes, B. A. A., Mendonca, M., Nguyen, X.-N., Obraczka, K., and Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys Tutorials*, 16(3):1617–1634.

Open Networking Foundation (2023). P4Language Repository. `https://github.com/p4lang`.

Sánchez, P. M. S., Valero, J. M. J., Celdrán, A. H., Bovet, G., Pérez, M. G., and Pérez, G. M. (2021). A survey on device behavior fingerprinting: Data sources, techniques, application scenarios, and datasets. *IEEE Communications Surveys Tutorials*, 23(2):1048–1077.

Tan, L., Su, W., Zhang, W., Lv, J., Zhang, Z., Miao, J., Liu, X., and Li, N. (2021). In-band Network Telemetry: A Survey. *Computer Networks*, 186:107763.

Tu, N. V., Hyun, J., Kim, G. Y., Yoo, J.-H., and Hong, J. W.-K. (2018). Intcollector: A high-performance collector for in-band network telemetry. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 10–18.