# FEVER: Intelligent Behavioral Fingerprinting for Anomaly Detection in P4-based Programmable Networks

Matheus Saueressig[1], Muriel F. Franco[1], Eder J. Scheid[1], Alberto Huertas[2], Gerome Bovet[3], Burkhard Stiller[2], Lisandro Z. Granville[1]

**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.

## 1 Introduction

Society and companies' increasing demands for high-speed and reliable communication propel the evolution of computer networks. Programmable networks have emerged as a pivotal approach to these evolving requirements. This approach provides the flexibility and adaptability necessary to accommodate the diverse services and applications that enable the evolution of communications [13], thus helping to avoid network ossification. Examples of programmable networks and facilitating technologies include the concepts of Software-defined Networking (SDN) [1] and Network Functions Virtualization (NFV) [3].

[1]Institute of Informatics (INF) – Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil
E-mail: {msaueressig|mffranco|ejscheid|granville}@inf.ufrgs.br
[2]Communication Systems Group CSG, Department of Informatics IfI – University of Zurich UZH
Zürich, Switzerland
E-mail: {huertas|stiller}@ifi.uzh.ch
[3]Cyber-Defence Campus, armasuisse Science & Technology
Thun, Switzerland
E-mail: gerome.bovet@armasuisse.ch

Programmable networks allow behaviors and services to be changed rapidly and fashionably. SDN improves network management by decoupling the control plane from the data plane, thus building more flexible and efficient networks by running intelligent controllers out of the switches [1]. However, SDN is still dependent on protocols like OpenFlow, which might be negatively impacted by the different implementations of data path hardware that vary according to vendors. This makes complex the management of different types of switches [7]. There are also approaches emerging as an ally to add programmability for the data plane. The Programming Protocol-independent Packet Processors (P4) is a protocol and vendor-independent solution [4] that defines new packet processing protocols without needing specific hardware support. It enables the development of protocols tailored to specific network requirements. Also, it allows the monitoring of insightful metrics using frameworks that allow for collecting and reporting network states by only using the data plane [16].

One critical aspect of computer networks is cybersecurity, which is not an exception for programmable networks and P4 implementations. Currently, there are research efforts towards security solutions to detect and mitigate cyberattacks, such as P4-based firewalls [17] and detection of Distributed Denial-of-Service (DDoS) using Machine Learning (ML) [12]. Also, there are considerable efforts for network verification based on assertions to identify faulty P4 programs [19] and bug-free programs [6]. Although such solutions are promising, the existing solutions focus on network traffic and targeting specific cyberattacks, thus not covering the detection of possible malicious behaviors from the switch and P4 program perspective. Therefore, there is still room for novel approaches correlating vast amounts of data and metrics available for a fine-grained analysis of networks and P4 program behaviors during run-time, thus allowing for detecting malicious behavior in networks considering both traffic and P4 programs running. Behavioral fingerprinting can be an ally for such analysis as it can provide a deeper understanding of device behavior beyond traditional monitoring metrics. Examples of scenarios where behavioral fingerprinting was successfully applied include the detection of ransomware in resource-constrained devices [15] and, on programmable networks, for identifying operating systems running on hosts [2].

Therefore, we advocate that intelligent behavioral fingerprinting [15] can be used to model patterns and characteristics of network devices and P4 applications during routine operation. After that, patterns could be used to identify anomalies in the traffic of P4 programmable switches or behavior changes during the execution of a Firewall developed as a P4 program, among others. We assume that each device and program may exhibit unique traffic flow characteristics, protocol usage, resource consumption, and responses to varying network conditions. By capturing and analyzing such unique behavioral patterns, behavioral fingerprinting can offer a more comprehensive and context-aware perspective for network management activities. Based on that, it is possible to identify malicious behaviors on programmable networks even before they become a more complex problem, including cyberattacks at early phases and malicious changes in P4 programs.

Thus, in this work, we propose FEVER, a framework for behavioral fingerprinting of programmable switches running P4 applications to identify anomalies by combining resource consumption and network telemetry. For that, the framework *(i)* identifies a set of metrics (*e.g.*, network-centric and resources consumption) to be used for the behavioral fingerprinting of programmable networks and *(ii)* describes a clear path to generate the fingerprints. Furthermore, the FEVER framework consists of two unsupervised ML algorithms [18] implemented and trained to identify anomalous traffic (*e.g.*, elephant flows and DDoS) coming from different hosts and modifications on P4 program code that change their behaviors. The feasibility and performance of FEVER is evaluated in a realistic virtualized scenario composed of virtual P4-based switches and P4 programs. The proposed ML algorithms are evaluated based on the F1-score and application scenarios emulated using Mininet and bmv2 switches, followed by a discussion of key findings. The results show that our frameworks can identify different P4 program behaviors and traffic overload in specific switches by looking at the behavior fingerprinting generated based on switches' resource consumption.

The rest of this document is organized as follows: Section 2 presents related work. The FEVER framework is described in Section 3, followed by evaluation and discussion in Section 4. Finally, in Section 5, the conclusion and future work are presented.

## 2 Related Work

We conducted a systematic literature, particularly of applications for detecting and mitigating cyberattacks and managing cybersecurity in programmable networks. Work focused on three aspects was identified: *(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 [9], identifying DDoS attacks by statistically analyzing traffic flows [5], and also verifying P4 programs using static checks [6] and assertions [19] to identify possible execution faults. Furthermore, P4 has also been used to identify devices' Operational Systems (OS) and react to them accordingly, such as dropping packets or defining rate-limit for specific OS types [2]. 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 (*i.e.*, 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 [8]. 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 [18]. 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 [11], 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) [15], is still is underinvestigated in the context of programmable network security.

## 3 The FEVER framework

FEVER is proposed to explore programmable device behavior fingerprinting for misbehavior detection in programmable switches, traffic, and network programs running as part of the network (*i.e.*, P4-based programs). 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 FEVER architecture, with the different components and steps described. The architecture is divided into three modules: *(i)* Data Analysis, which represents the tasks related to the definition of a possible range of metrics and also processing collected data, *(ii)* Behavior Fingerprinting, which determines the scenarios to be considered (*e.g.*, normal and anomalous) and train the models to identify the different behaviors of P4 programs and devices; and, finally, the *(iii)* Testbed represents the environment used to monitor the behaviors in order to create datasets for behavior fingerprinting and also detect anomalies. Each of these modules and their respective components are described below.

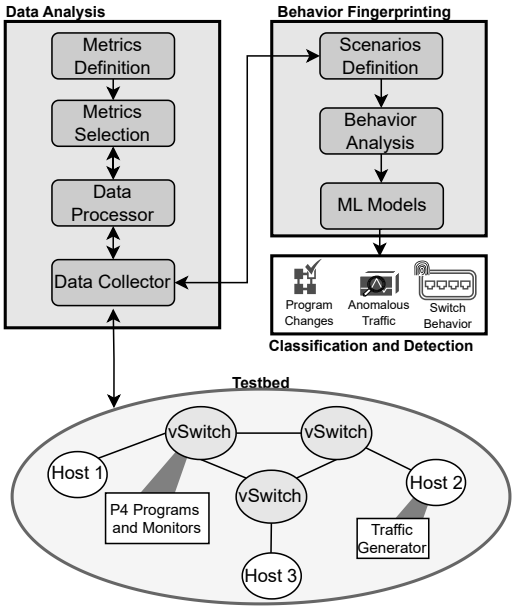


**Fig. 1** FEVER Architecture

In the *Data Analysis*, the first step consists of defining candidate behavioral metrics to be monitored, such as memory and Central Processing Unit (CPU)-related resources, In-band network telemetry framework, and network policies. Such metrics might change according to the environment in which the programmable network is built (*e.g.*, Intel Tofino or a Mininet-based emulation). After defining metrics, in the next step, it is necessary to select those that are relevant to detect different behaviors. For that, applying different techniques to identify the correlation and dependence between metrics is possible. Possible techniques include heatmaps to understand the relationship between metrics and linear plots to verify the behavior of two or more metrics under regular and anomalous circumstances.

The tasks involved in the collection and processing of data are also part of the *Data Analysis* module. The *Data Processor* is a component that receives data collected by the *Data Collector* and refines the data to make it more digestible by the *Metrics Selection* and other modules. Such refinements include removing noise from the raw data and preparing to plot the results for behavior analysis. The *Data Collector* also communicates with the *Behavior Fingerprinting* module to to collect data from the monitors and whether additional data processing is needed.

For the FEVER framework, we consider both in-device and externally collected behavior sources. Examples of metrics mapped and being considered in our implementation are shown in 1. The resource consumption is divided in terms of CPU and Random-Access Memory (RAM), which also include different levels of granularity, such as CPU migrations, Instructions and cycles, page faults, and Resident Set

Size (RSS). Such metrics allow us to understand behavior better and generate the fingerprints. Additional metrics can also be considered, including Ternary Content Addressable Memory (TCAM), a high-speed memory widely used in networking devices and programmable switches. To collect resource data from the switches, we have written a shell script that runs the Linux *perf* and *proc* commands simultaneously, while *iperf* was integrated into the script to create different flow behaviors according to the test needs, such as a normal flow of packets between all switches and specific elephant flows.

**Table 1** Overview of Metrics Considered for Behavioral Fingerprinting

| Metric | Example 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. | proc |
| RAM | RAM monitoring helps us to detect anomalies if they allocate memory to do malicious activity or due to misconfiguration. | perf |
| 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 |
| Syscalls | Analysis of additional events running in background can improve the detection performance | perf |
| Processes | It is important to identify processes that represents virtual switches or that perform core functions | ps aux |
| Packet Header | Analysis of per-packet headers to statistically characterize switch behavior | INT and Wireshark |

Figure 2 shows an example of collected metrics selection using a heatmap. 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 high correlation can be maintained since they did not correlate to other values more than once and due to the high granularity of the values. After the heatmap analysis, a line plot can be provided for a more in-depth analysis of overlapping metrics.

The fingerprinting is finally built in the last module. For that, the scenarios are defined to determine which behaviors have to be considered for the generation of the training dataset and traffic generation. For example, P4 program actions and behaviors can be monitored to identify program changes and bugs, while switch resource consumption can be analyzed to understand anomalous behavior. The *ML Models* are then generated to identify program changes, abnormal traffic before it disrupts the network, and anomalous detection by monitoring the behaviors of network switches. Different ML techniques can be used for behavioral fingerprinting because they can detect complex data patterns and handle multi-variate data.

An instance of the FEVER is implemented as a proof-of-concept, running on Mininet and providing ML models based on One-Class Support Vector Machine (OCSVM) and Local Outlier Factor (LOF) as unsupervised learning for behavior fingerprinting. These ML techniques were used because of their simplicity and potential to identify outliers based on normal behaviors. Also, scripts were implemented for monitoring virtual switches and generating traffic according to specific scenarios. The source-code of the FEVER and datasets are publicly available at [10].

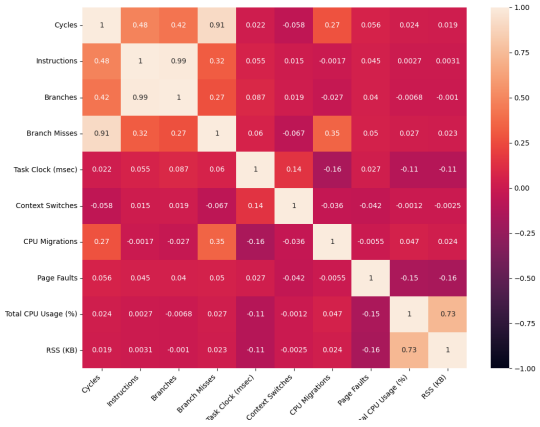


**Fig. 2** Heatmap for Metrics Correlation and Selection

## 4 Evaluation

To validate our approach, we have developed a data pipeline focused on monitoring how anomalies (*e.g.*, unexpected traffic and changes on P4 programs) alter the CPU and memory usage in individual switch devices and compared this situation with a regular data flow. For this experiment, we have created a fingerprint of P4-enabled switch considering the CPU and memory metrics, thus, analyzing the resources usage of processes related to bmv2 while running a Mininet network. We run a topology of three switches and three hosts, configured as the testbed shown in Figure 1. All the switches run a P4 program called Multi-Hop Route Inspection (MRI), which enables users to monitor the routes taken by packets and their associated queue lengths. A Python script was developed to set up the Mininet environment and configure the hosts and bmv2 switches. As described in the framework, *iperf* was also used to create the traffic flows to simulate behaviors.

The monitoring scripts implemented by FEVER collect metrics each second and save the collected data to a Comma-Separated Values (CSV) file for each switch for one hour. Each virtual switch's Process ID (PID) is identified for that. As Mininet treats each switch as a single process, it is possible to identify the PID corresponding to the switch activity and monitor it using *proc* and *perf* commands. Thus, after one hour of running the simulation, we gathered the monitored data from CSV files generated with the PID of the switches being monitored. Each experiment consisted of running the desired P4 program (from the p4lang tutorial repository) [14] for one hour with modifications to generate the desired traffic.

The behaviors considered consist of *(i)* normal traffic, *(ii)* anomalous traffic, and *(iii)* modified P4 behavior. Normal traffic was defined as simple requests that use less than 10% of network capacity sent from two clients to a server. The *(ii)* anomalous traffic is a high-traffic flow in which the hosts flood the network using the maxi-

mum available throughput. For the *(iii)* modified behavior, we consider an MRI with modifications, such as a conditional branch and arithmetic operations.

The experiments have to be run 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, such as all metrics with 99% of correlation identified using heatmaps (*cf*. Figure 2). Next, a manual analysis is performed to verify potential clear behaviors and outliers, thus, allowing the human in the loop to calibrate the model and understand its feasibility. In case of simple scenarios, a basic set of rules can be defined, but when this manual verification shows complex correlations, our ML models must be used. We ran our experiments three times for each behavior to create our training dataset for normal, anomalous, and modified behaviors. Therefore, the experiments were run nine times, with a 1-hour duration each. The performance of the ML models is analyzed using the well-established metrics *recall, precision, and F1-score*. All features were selected and extracted following the FEVER framework as introduced in Section 3. We used 3,240 samples of normal behavior for training and 6,840 samples of all behaviors (normal, anomalous traffic, and modified P4 program) for evaluation.

### *4.1 Detection of Anomalous Traffic*

This experiment shows the performance of FEVER to detect abnormal data flow increases through a switch to identify possible flooding attacks earlier, such as Ping Flood, SYN/ACK Flood, and HTTP/HTTPS Flood. We have used the *iperf* tool to create anomalous traffic 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, *i.e*., it floods the network with UDP packets to measure the maximum throughput. We have considered this scenario ideal since it would not disrupt the network but would send enough traffic to be detected. In a real-world scenario, detecting an increase in flow before a DDoS caused by the data flood would be beneficial.

Both Host 1 and Host 2 behave like clients, and Host 3 is the server. Clients send UDP packets to the service, creating a high-traffic flow through Switch 3, the one connected to Host 3. Since the high-traffic flow was directed to Host 3, we analyzed only Switch 3 to detect the anomaly. The RSS was identified as the best metric for such a scenario based on initial analysis (*e.g*., heatmaps, overlapping, and line plots) of metrics collected. After the heatmap analysis, line plots were provided to identify metrics with overlapping or unidentifiable anomalous behavior, which have to be automatically identified by ML algorithms. The dropping of such overlapping increases the accuracy of detection algorithms, but it is not a realistic task in real environments since abnormal behavior is unknown in real environments. Therefore, the ML models must be used to understand metrics even when overlapping happens.

Figure 3 shows the overall performance of both ML models implemented using different features for anomalous traffic identification, including all features combined. Even though RSS alone has a maximum performance, it is dangerous to rely on it alone since this could indicate an overfitting of the model. Anomaly detection relies on robustness and identifying unknown patterns incompatible with normal be-

havior. Using only one feature might incapacitate the ability of different scenarios to be analyzed, and it generates trained models for specific situations rather than a generalized approach. Using all features has a high F1-Score, so it would not compromise the overall analysis, and it would guarantee the possibility of more data from unknown patterns being collected, thus identifying anomalies.

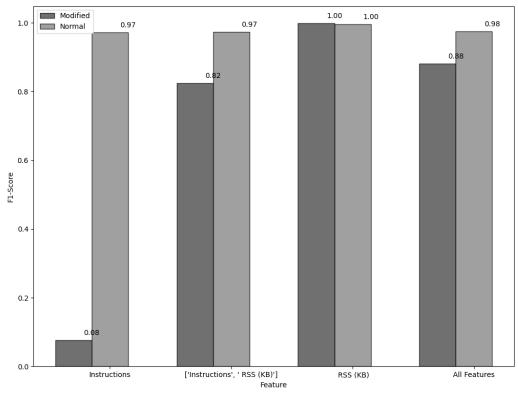


**Fig. 3** F1-score for the Detection of Anomalous Traffic using Different Features

The OCSVM achieved an maximum accuracy for identifying anomalous behavior in terms of high traffic in a determined switch. Since it is an ML algorithm designed for situations where only examples of the normal class are available during the training phase, it learns a boundary around the regular instances. It classifies any value outside this boundary as an anomaly. The magnitude difference between the RSS memory and lack of overlapping helped the algorithm ideally detect the anomaly. However, relying only on features high relevance by dropping other features might overfitting the model since the nature of an anomaly is unknown. OCSVM have learned a boundary that effectively captures this separation. This can be verified if we drop the RSS feature (the most relevant metric). In this scenario, OCSVM performs poorly, with a recall of 0.43 and a precision rate of 0.56.

The LOF also scored the same as the OCSVM, thus, achieving a perfect accuracy for scenarios where the anomaly is known. LOF, in particular, is designed to identify local deviations from most data points. Therefore, if anomalies form distinct clusters or have noticeable local differences, LOF can excel in detecting them. As we have isolated regions detected by the LOF, the overlapping did not interfere in the model performance. In conclusion of such an experiment, memory analysis is shown to be a reliable source to detect flow changes and, thus, interesting to prevent flood attacks or network disruptions since, with more packets being processed, more memory needs to be allocated to operate the switch processes properly.

### *4.2 Detection of Changes on P4 Programs*

We also created behavior fingerprinting for two scenarios: *(i)* when the same program is running but with possible malicious modifications and *(ii)* when a different program is running (*i.e.*, program identification). Figure 4 shows the overall F1-score for identifying changes in a P4 program (*i.e.*, original MRI but with a basic math operation and new variables included in the source-code). The F1-score is provided for both LOF and OCSVM using different features (*e.g.*, CPU instructions and RSS combinations) to identify if the running program is correct (normal) or modified according to scenario *(i)*. It shows that the RSS still plays an important role when identifying modified behavior, where CPU instructions can be used as an ally to identify normal behavior.

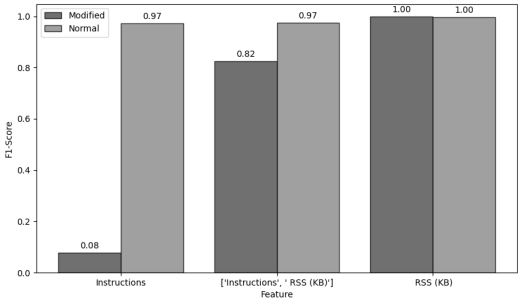


**Fig. 4** F1-score for the Detection of Changes in the Original P4 Implementation of the MRI

Besides that, we have checked if the behavioral fingerprinting can also identify which program is running. For that, different metrics can be used since the program's resource consumption vary according to their functionalities and processing demands. Figure 5 shows that each metric's percentage varies according to a pre-defined baseline. For that, we have used the resource consumption of a *Basic Forward* as a baseline and analyzed how much (in terms of percentage) each of the following P4 programs surpasses the baseline resource usage: *MRI, Link Monitor, and Explicit Congestion Notification (ECN)*. All P4 programs, including the baseline, are the same as those available at the p4lang repository [14].

As can be seen, all metrics vary for the programs, especially those related to memory and CPU usage. This can be used then to create a fingerprint to identify specific programs running in the network and also highlight if malicious changes are made in the switch, such as when a malicious operator replaces P4 programs to affect the network service chain.

By using such metrics to determine the behavior fingerprint of each P4 program, we were able to classify 100% the correct problem, thus identifying what is running in the switch by only looking at the resource consumption. When dropping some metrics (*e.g.*, RSS, page faults, and CPU migrations), our model achieves a performance of 95% accuracy in identifying the correct program running.

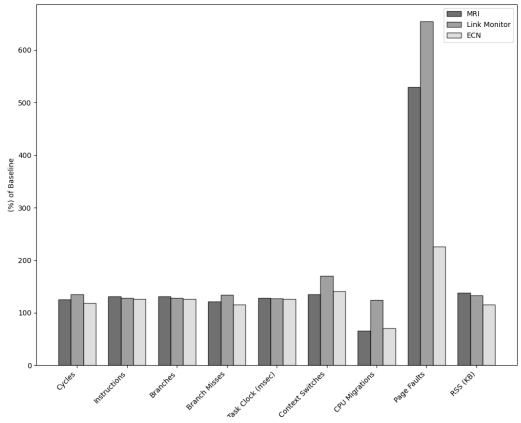


**Fig. 5** Analysis of Consumption Behaviors of Different P4 Programs

## 5 Conclusions and Future Work

This work 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 are employed together with statistical processing to map and understand metrics that highlight potential anomalies or even normal behaviors given a given 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.

In conclusion, our experiments provide essential insights into the fingerprinting of programmable networks, and our FEVER framework has proven to be a consistent methodology for analyzing the behavior of programmable switches and P4 programs, which can be adapted to real-life scenarios. 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.

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. Furthermore, implementation on real-world scenarios composed by Tofino switches is envisioned.

## References


1. S. Badotra, S. N. Panda: Software-Defined Networking: A Novel Approach to Networks. Handbook of Computer Networks and Cyber Security: Principles and Paradigms pp. 313–339, 2020
2. S. Bai, H. Kim, J. Rexford: Passive OS Fingerprinting on Commodity Switches. In: IEEE 8th International Conference on Network Softwarization (NetSoft), 2022, pp. 264–268
3. Bondan, L., et al.: FENDE: Marketplace-Based Distribution, Execution, and Life Cycle Management of VNFs. IEEE Communications Magazine **57**(1), 13–19, January 2019
4. P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker: P4: Programming protocol-independent packet processors. SIGCOMM Comput. Commun. Rev. **44**(3), 87–95, jul 2014
5. D. Ding, M. Savi, D. Siracusa: Tracking Normalized Network Traffic Entropy to Detect DDoS Attacks in P4. IEEE Transactions on Dependable and Secure Computing **19**(6), 4019–4031, 2021
6. D. Dumitrescu, R. Stoenescu, L. Negreanu, C. Raiciu: bf4: Towards Bug-Free P4 Programs. In: SIGCOMM 2020. Virtually, USA, 2020, pp. 571–585
7. B. Goswami, M. Kulkarni, J. Paulose: A survey on p4 challenges in software defined networks: P4 programming. IEEE Access , 2023
8. F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, M. Menth: A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research. Journal of Network and Computer Applications **212**, 103561, 2023
9. G. Li, M. Zhang, C. Liu, X. Kong, A. Chen, G. Gu, H. Duan: NETHCF: Enabling Line-Rate and Adaptive Spoofed IP Traffic Filtering. In: IEEE 27th International Conference on Network Protocols (ICNP 2019). Chicago, USA, 2019, pp. 1–12
10. M. Saueressig, M. F. Franco: FEVER-P4 Repository, 2024, `https://github.com/ComputerNetworks-UFRGS/FEVER-P4`
11. M. Saueressig, M. F. Franco, E. J. Scheid, L. Z. Granville: An Approach for Behavioral Fingerprinting of P4 Programmable Switches. In: XX Escola Regional de Redes de Computadores (ERRC 2023). Porto Alegre, Brazil, 2023, pp. 22–60
12. F. Musumeci, V. Ionata, F. Paolucci, F. Cugini, M. Tornatore: Machine-Learning-Assisted DDoS Attack Detection with P4 Language. In: IEEE International Conference on Communications (ICC 2020). Virtually, 2020, pp. 1–6
13. B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, T. Turletti: A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. IEEE Communications Surveys & Tutorials **16**(3), 1617–1634, 2014
14. Open Networking Foundation: P4Language, 2023, `https://github.com/p4lang`
15. P. M. S. Sánchez, J. M. J. Valero, A. H. Celdrán, G. Bovet, M. G. Pérez, G. M. Pérez: A survey on device behavior fingerprinting: Data sources, techniques, application scenarios, and datasets. IEEE Communications Surveys & Tutorials **23**(2), 1048–1077, 2021
16. L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, N. Li: In-band Network Telemetry: A Survey. Computer Networks **186**, 107763, 2021
17. L. Teng, C. H. Hung, C. H. P. Wen: P4SF: A High-Performance Stateful Firewall on Commodity P4-Programmable Switch. In: IEEE/IFIP Network Operations and Management Symposium (NOMS 2022). Budapeste, Hungary, 2022, pp. 1–5
18. M. Usama, J. Qadir, A. Raza, H. Arif, K. L. A. Yau, Y. Elkhatib, A. Hussain, A. Al-Fuqaha: Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges. IEEE Access **7**, 65579–65615, 2019
19. Q. Wang, M. Pan, S. Wang, R. Doenges, L. Beringer, A. W. Appel: Foundational verification of stateful p4 packet processing. In: 14th International Conference on Interactive Theorem Proving (ITP 2023). Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023