IoT Device Fingerprinting on Commodity Switches

Pulkit Garg
Department of Computer Science
Dalhousie University
pulkit.garg@dal.ca

Miguel Neves
Department of Computer Science
Dalhousie University
mg478789@dal.ca

Israat Haque
Department of Computer Science
Dalhousie University
is179864@dal.ca

***Abstract*—The number of IoT devices and the concept of smart homes have become prevalent these days. This paper presents FingerP4, a stateful solution that uses P4 programming language to uniquely identify IoT devices inside a smart home entirely in a BMv2 switch. FingerP4 uses packet lengths, the direction of flow, and the state of the packet defined by a Finite State Machine (FSM) to identify the devices. In our initial experiments, FingerP4 was successfully able to identify events from 7 different IoT devices entirely in the data plane.**

I. Introduction

With the advancement of technology in the 21st century, computers have become miniature and powerful in computation. Internet of Things (IoT) devices are a result of this rapid evolution of computers. As the name suggests, IoT devices are hardware objects embedded with sensors and actuators that are programmable and transfer data over the network [1]. A group of these devices interconnected through the Internet inside a home for various purposes such as temperature, lighting, accessing television, security controls, etc., forms a Smart Home [2]. Currently, there are 258.54 million active smart homes globally, which is expected to reach up to 482.8 million by 2025 [3].

Unfortunately, with this rising use of IoT devices and smart homes, several privacy and security concerns have emerged. Various solutions have been proposed to protect the home IoT devices. For example, Trimananda et al. proposed an intelligent system, PingPong, to uniquely identify (fingerprint) a home IoT device based on packet exchanges between the home device, the cloud, and the smartphone even when traffic is encrypted [4]. PingPong can extract unique signatures from a dataset containing device events such as ON/OFF. However, PingPong can suffer from performance degradation at scale due to the limited capacity of the server that processes the IoT traffic.

In recent years, developments in technology have also led to the invention of programmable networks. For instance, Software-defined Networking (SDN) [9] is a new networking paradigm that makes the network easy to configure and manage by separating the network intelligence (control plane) from data forwarding elements (data plane) to make the network control programmable [10][11]. The control plane manages how packets are forwarded, while the data plane is responsible for sending packets from the source to the destination [12].

The second generation of SDN takes network programmability one step ahead by directly programming data plane

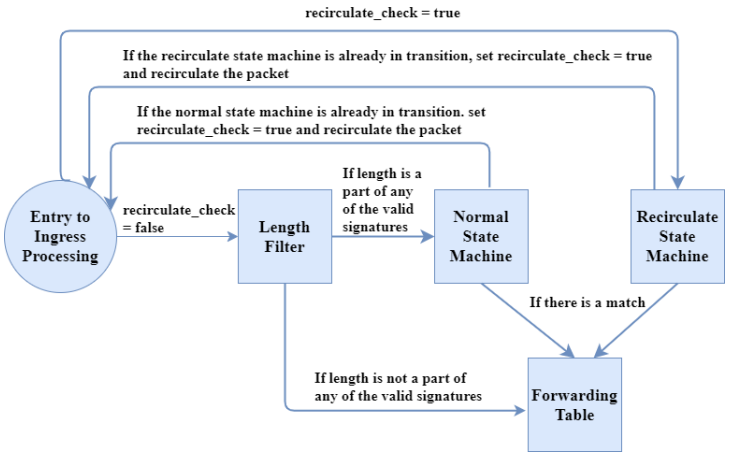

Fig. 1. Fingerprinting Design

switches using a new programming language called P4 [5]. Such programmability can allow a switch to process home IoT traffic at line rate (Tbps) without needing a server (middlebox). Also, switches naturally sit between the home IoT devices and the cloud servers.

In this poster, we propose a new fingerprinting system, FingerP4, built on PingPong to identify devices' signatures entirely on the data plane. We implement the proposed system in the Mininet emulator, where we program software switch BMv2 using P4 [5] to demonstrate the feasibility of FingerP4.

In a traditional SDN network, packets from the switch must be sent back and forth to the controller which leads to wastage of time and resources. Hence by transferring some of the features to the switch, we not only give more freedom to the programmer, but also are more efficient while following the principles of SDN [13].

Secondly, the invention of Programmable Independent Switch Architecture (PISA) led to development of high speed forwarding protocol independent switches, with forwarding speeds up to 6.4 Tb/s, which do not understand any protocols unless programmed [14]. To take advantage of these speeds, we utilize the processing capabilities of the switch by identifying devices and forwarding packets in the data plane. Additionally, our solution could be used by Internet Service Providers (ISP) to make their network secure by fingerprinting the IoT devices in their network and help monitor if an unidentified malicious device tries to connect to the network.

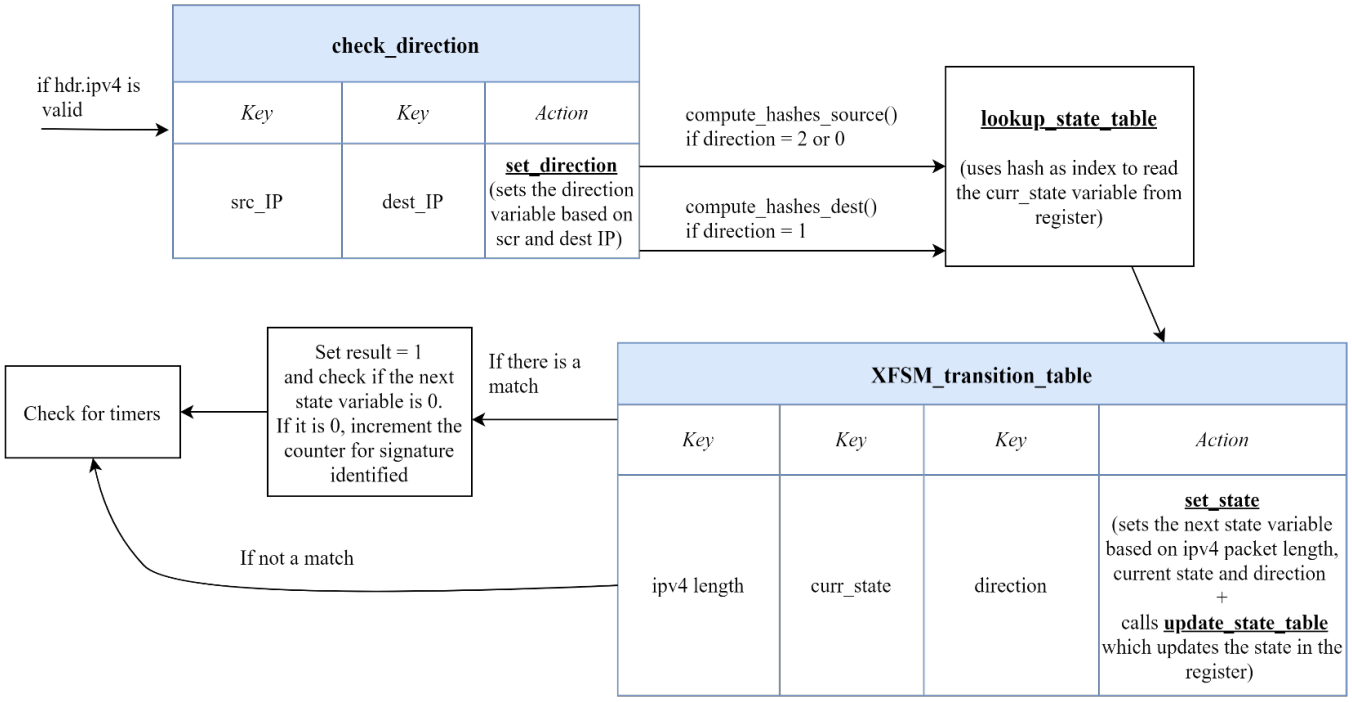

Fig. 2. State Machine Function

The knowledge of all the devices connected to the network could help in mitigating security threats.

Related Work. Bianchi et al. use eXtended Finite State Machines (XFSM) and P4 to present a platform agnostic solution by programming the switches with stateful packet processing capabilities [13]. Additionally, there have been efforts to identify the Operating System of the devices in the data plane using P4 [6] and identifying events in IoT devices uniquely in the control plane using Java [4]. We identified events in an IoT device entirely in the data plane using P4 and the concepts of Finite State Machine (FSM).

II. Identifying devices in the data plane

A. Overview

This project takes up the concepts of PingPong and applies it to a stateful solution in P4 programming language to identify devices inside a BMv2 switch. For the switch to identify signatures from a network traffic, the signatures have to be stored in the switch. Initially signature text files for each device are generated using the PingPong software. Python scripts then extract relevant signature information from these files which is then converted into JSON file to configure the switch. The P4 program is then installed on the switch and the match-action tables are configured using forwarding rules based on the JSON files. Afterwards, the network traffic is replayed on a switch interface to test FingerP4.

B. Data Plane Design

The State Machine abstraction is utilized in identifying a signature in the fingerprinting design. Figure 1 gives a brief overview of how a packet is processed in the Ingress processing. Since all the packet processing for FingerP4 occurs before the output port for the packet is set, the whole design is implemented inside ingress pipeline [8]. Normal State Machine and Recirculate State Machine are the two instances of the State Machine Function (II - C). We use a metadata variable, recirculate_check, to check if the packet being processed is a recirculated packet. It is then passed through a length filter which checks if the Ipv4 length of the packet matches any of the lengths from all the valid signatures. The length filter reduces the number of false positives and the load on our design by only allowing the relevant packets to access the state machines. If a packet matches the length filter, it is sent to the Normal State Machine. In case the Normal State Machine is handling another signature, the packet is recirculated and the recirculate metadata is set to true. Recirculated packets are sent to the start of the ingress pipeline for re-processing. Recirculated packets are then sent to the recirculate state machine for processing. Therefore, a packet from a valid signature would always find a match either in the recirculate state machine or in the normal state machine. As soon as the signature is found, the packet is forwarded to the destination port using a forwarding table.

C. The State Machine Function

P4 programming language is used along with an FSM mechanism to achieve the goal of identifying signatures of devices in the data plane. To understand the design for the project, it is important to explain the FSM used in the solution. Figure 2 gives a brief overview of how our state machine

function works. After checking if the ipv4 header in the packet is valid, we use a match action table to set the packet direction. A CRC32 function is applied on the source or destination IP depending on the direction of travel. This hash function is used as an index for registers to retrieve the current FSM state (lookup_state_table). The retrieved state along with packet length and direction is used in a match action table to find the next state (XFSM_transition_table). On a table hit the next state is stored in a variable used to update the state in the register for the associated hash index. If the next state variable is set to 0, the signature has been fully identified. If there is a match, then a return variable is set to 1 and the value of next state is checked to see if a signature is fully identified. The packet's timestamp is then compared with a timeout value to check if the packet lies in the duration of the event in the IoT device for which the signature is being identified.

III. Preliminary Results

We used a BMv2 switch on a virtual machine provided by the p4 developers to test our solution. The PCAP files provided by the PingPong researchers were used to replay the traffic using tcpreplay [8] on the virtual switch in a mininet environment with two hosts.

We used several datasets from distinct IoT devices individually to test our solution. Table 1 shows the results for signatures identified by PingPong and signatures identified by FingerP4. We realized that BMv2 drops some packets at high traffic rates, so the packets were replayed at a rate of 100 packet per second to get an accurate idea of system behaviour.

TABLE I
SINGATURES IDENTIFIED

Device Name	PingPong	FingerP4
Amazon Plug	99	99
Ecobee Thermostat Fan	100	100
TP-Link Bulb (On/Off)	100	100
Wemo Plug	100	100
Ring Alarm	98	97
D-Link Plug	101	101
Sengled Bulb (ON/OFF)	97	96

As we can see from the table, most of the signatures identified by our system match the number identified by PingPong. There are minor differences which could be attributed to the behavior of Bmv2 switch. For certain pcap files, the switch could become congested with the number of packets, dropping some of them and lowering the number of signatures detected. We might be able to solve it with either decreasing the rate of packets or increasing the size of buffers inside the switch.

IV. Conclusion and Future Work

In this paper, we presented a tool called FingerP4, programmed in the P4 programming language, which can identify IoT devices in the bmv2 switch. By doing it on the data plane we can reduce the load on the control plane and also utilize the processing power of the switch. To achieve this goal, we used PingPong to generate signature files to uniquely identify devices and events, ON/OFF, based on packet lengths and direction of flow. Future work includes testing our solution on a real hardware switch and evaluating the results. Also, additional testing parameters such as throughput analysis could be introduced to compare if detecting signatures in the data plane is more efficient than in a server based version. Datasets from more devices provided by PingPong researchers could also be used to test a greater variety of devices.

References

[1] A. Ltd., "What are IoT Devices", Arm — The Architecture for the Digital World, 2021. [Online]. Available: https://www.arm.com/glossary/iot-devices.

[2] J. Chen, "Smart Home", Investopedia, 2021. [Online]. Available: https://www.investopedia.com/terms/s/smart-home.asp.

[3] A. Holst, "Topic: Smart home", Statista, 2021. [Online]. Available: https://www.statista.com/topics/2430/smart-homes/.

[4] R. Trimananda, J. Varmarken, A. Markopoulou, B. Demsky, "Packet-Level Signatures for Smart Home Devices," in Proceedings of the 2020 Network and Distributed System Security Symposium (NDSS). February 2020, San Diego, CA.

[5] A. Fingerhut, A. Bas, A. Sivaraman and D. Arora, "p4lang/behavioral-model", GitHub, 2020. [Online]. Available: https://github.com/p4lang/behavioral-model.

[6] S. Bai, H. Kim, and J. Rexford, "Passive OS Fingerprinting on Commodity Switches," 2019.

[7] Klassen, F., 2021. Tcpreplay - Pcap editing and replaying utilities. [online] Tcpreplay.appneta.com. Available at: https://tcpreplay.appneta.com/

[8] OpenFlow Switch Specification - Open Networking Foundation. Open Networking Foundation (ONF), 2014, p. 35.

[9] C. Talk, "What is Software-Defined Networking (SDN)? - Ciena", Ciena.com, 2021. [Online]. Available: https://www.ciena.com/insights/what-is/What-Is-SDN.html.

[10] "The basics of SDN and the OpenFlow Network Architecture", NoviFlow, 2021. [Online]. Available: https://noviflow.com/the-basics-of-sdn-and-the-openflow-network-architecture/.

[11] C. Craven, "What Is OpenFlow? Definition and How it Relates to SDN", sdxcentral, 2021. [Online]. Available: https://www.sdxcentral.com/networking/sdn/definitions/what-is-openflow/.

[12] S. Jena, "Difference between Control Plane and Data Plane," GeeksforGeeks, 18-Aug-2020. [Online]. Available: https://www.geeksforgeeks.org/difference-between-control-plane-and-data-plane/. [Accessed: 21-Jul-2021].

[13] OpenState: G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch" ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, pp. 44–51, 2014

[14] C. Kim, "The Forwarding Plane: An Old New Frontier of Networking Research," in CS244, 21-Jul-2021.