IoT Device Fingerprinting on Commodity Switches

Carson Kuzniar*, Miguel Neves*, Vladimir Gurevich†, Israat Haque*
Dalhousie University*, Intel BXD†

***Abstract*—IoT devices such as wearables, voice assistants and home appliances are becoming an integral part of our lives. However, these devices still represent a security and privacy risk with large-scale coordinated attacks often populating the news. The ability to tell which IoT devices are where in a network (i.e., to fingerprint them) can help administrators to mitigate such attacks at the earliest stages. While fingerprinting solutions exist, they often work offline, depend on sampled data or rely on payload information to work. In this paper, we propose PoirIoT, a high-speed in-network system for fingerprinting IoT devices. PoirIoT is based only on packet metadata (e.g., length and direction) and can detect a device as soon as it exchanges its first packets. We implement a prototype of PoirIoT on a Tofino-based programmable switch and show it can detect *all* possible IoT devices on a publicly available dataset. Moreover, PoirIoT runs at line rate and incurs minimal resource overhead on the programmable switch ASIC.**

I. Introduction

Smart homes are becoming increasingly popular due to the proliferation of IoT devices and their associated applications. For instance, 33 million North American houses currently have a smart thermostat to efficiently utilize energy [1]. Also, the number of active IoT devices is expected to reach 30 billion by 2025 [2]. While smart home devices are mostly connected to a cloud backend for monitoring and remote control, their network connectivity can enable malicious actors to compromise them and launch large-scale coordinated attacks. As an example, the Mirai botnet could successfully compromise more than 100K IoT devices and engage them in coordinated DDoS attacks over the last few years [3]. In addition, home-based IoT devices have recently raised numerous privacy concerns [4].

Existing solutions for securing IoT devices at scale invariably involve fingerprinting them (i.e., identifying the device type, manufacturer and/or event) through different forms of traffic analysis. Previous work has focused on inspecting packet payloads, calculating traffic statistics [5], and applying machine learning algorithms [6] to detect a particular device. In common, all these efforts run offline (i.e., based on collected traces) and require an external compute node to process data. Unfortunately, this approach faces important issues in terms of accuracy and speed when dealing with high traffic rates (above 10 Gbps), which are becoming the norm on current networks.

The emergence of programmable switches (e.g., Intel Tofino) and their high packet processing capabilities (up to Tbps) opens up an unprecedented opportunity to fingerprint IoT devices directly in the data plane while traffic passes through an ISP. ISPs are naturally situated in-between IoT devices and the cloud backends, and their switches represent a sweet spot to deploy novel security functions without introducing much overhead (server-based approaches require extra bandwidth for collecting traffic and beefy processors to analyze it). In addition, a single switch can serve multiple homes, which easily enables deploying a fingerprinting solution at scale compared to home-based alternatives. Ultimately, network administrators can use information about detected IoT devices to quickly react to threats and/or take preventive measures. For example, they can tweak firewall or access control rules, configure intrusion detection systems, or reach out to clients for patching known vulnerabilities as soon as they know the nature of the subscribed devices.

To realize this view, we propose PoirIoT, a high-speed in-network system for fingerprinting IoT devices. PoirIoT relies only on packet metadata (e.g., length and direction) to identify a device and can run at line rate on currently available switches. The system has two major components: a controller and a data plane module. The controller is primarily responsible for extracting signatures from packet traces and generating configuration rules for PoirIoT's data plane. The data plane, on the other hand, deploys a finite-state machine (FSM) abstraction to track relevant sequences of packets while looking for signature matches. We implement a prototype of PoirIoT in P4 [7] and evaluate it on an Intel Tofino switch [8]. We run our system over a publicly available dataset containing traces from fourteen popular home IoT devices, including smart plugs, light bulbs, thermostats, and home security systems. Our results show that PoirIoT can accurately detect *all* IoT devices while incurring minimal resource overhead on the programmable switch ASIC. In addition, PoirIoT can also detect most (above 70%) of the events from the detected devices. In summary, this paper makes the following contributions:

- We present the design and implementation of PoirIoT, an in-network system for fingerprinting IoT devices in high-speed networks.
- We conduct a thorough evaluation of PoirIoT using a real-world dataset and demonstrate it achieves 100% device detection accuracy. As a bonus, we also show PoirIoT can detect the majority of device events.
- We make PoirIoT's code open source to support reproducibility and encourage further research. Our code is available at [9] under an Apache Licence.

The rest of this paper is organized as follows. Section II motivates moving IoT device fingerprinting to programmable network devices as a feasible solution for high-speed networks. Section III highlights previous work. The next section presents PoirIoT's design in detail. Section V describes our prototype

implementation and evaluation results. We discuss some future efforts in Section VI followed by conclusions.

II. Motivation

Existing systems face several challenges when attempting to detect IoT devices. In this section, we detail some of these challenges and the benefits a switch-based solution brings to this scenario.

Volume. Many IoT device fingerprinting systems (e.g., [10] [5] [11]) rely on dedicated servers for device detection, either online or based on full packet captures. These solutions, however, cannot keep up with the large amounts of traffic on high-speed links (which often support 10 Gbps nowadays [12]). As a result, they either face a significant drop in accuracy or take a long time (sometimes hours or even days) to detect a device. An alternative would be to set up detection mechanisms directly on user homes (e.g., by tweaking their home routers), but that quickly leads to management issues. For instance, providers would need efficient ways to ensure that detection rules are always up-to-date. Fingerprinting IoT devices in the switch data plane removes these speed constraints as the former can operate at line rate.

Granularity. Large volumes of traffic frequently force network operators to work on highly aggregate and sparsely sampled data (e.g., NetFlow, IPFIX traces) for further inspection [5] [13]. Such coarse-granularity is specially harmful for IoT device detection as many IoT devices (e.g., smart bulbs, plugs) are silent almost all the time and only send a few packets when actively used (e.g., turned on/off) [5]. Programmable switches can naturally inspect every packet as part of their forwarding process, so they are able to observe even the shortest IoT events.

Complexity. Lastly, current IoT device fingerprinting approaches normally adopt techniques that are based on heavy-computation methods (e.g., parsing payload information [14], computing higher-order statistics [15], applying machine learning inference [6] [16]). That not only hinders their application in widespread scenarios (e.g., payload-based approaches cannot process encrypted traffic), but also makes it very challenging to implement them on high-speed programmable switches. Thus, we propose a system to overcome the aforementioned challenges and accurately detect IoT devices in high-speed networks.

III. Related work

Our work integrates ideas from both IoT device fingerprint efforts and the growing area of in-network computing. Below we detail other studies considered in the development of PoirIoT.

IoT device fingerprinting. Prior work on IoT device fingerprinting utilizes a diverse range of techniques to detect IoT devices in the wild. For example, Peek-a-Boo [6] can detect devices in WLANs by sniffing wireless signals and applying machine learning algorithms over calculated statistics. HomeSnitch [16] takes a similar approach, but assumes access to a home router. Bremler-Barr et. al. [17] also position their work on a home router, but include DHCP features alongside traffic statistics in their classifier. Wifi Inspector [18] runs locally on the user's personal computer and uses expert rules in addition to machine learning for identifying scanned devices. IoTInspector [10] also takes advantage of personal computers to run, but instead requires users to label collected traffic which is further validated using a set of heuristics. In common, all these approaches are hard to manage at scale and provide limited insight from a network-wide perspective.

Afek *et. al.* [19] expand the perspective by deploying a hybrid model using a device at the customer's location and a virtual network function to monitor many home networks. Their detection relies on Manufacturer Usage Descriptions (MUD), so it is dependant on receiving up to date and correct information from manufacturers. Guo *et. al.* [14] and Saidi *et. al.* [5] propose new methods to detect devices based on packet captures at the ISP/IXP level. However, their methods either rely on payload information (e.g., DNS queries, TLS certificates) or are targeted at offline processing. PingPong [11] can identify devices using signatures that consist only of packet lengths and directions (similar to PoirIoT). Pinball [20] builds on these signatures by analyzing the probability distribution of lengths rather than order of arrival. Both approaches still requires mirroring traffic to a dedicated server for analysis. In contrast, PoirIoT can run directly on programmable switches online and at line rate.

In-network computing. Previous work has shown that programmable switches are effective at processing large volumes of information. Instead of detecting IoT devices, P40f [21] uses a programmable switch to infer a target operating system. Rather than packet exchanges, their inference is based on TCP signatures from a single packet. Meta4 [22] recognizes DNS requests and responses that pass through a programmable data plane. The authors explore DNS-based fingerprinting as a use case, but challenges still exist when dealing with encrypted traffic. PPS [23] demonstrates overcoming I/O bottlenecks using switches to perform a keyword (i.e., string) search. It scans a packet payload using DFAs, but risks packet loss under heavy loads because of recirculations. PoirIoT is complementary to these efforts and provides a new in-network application instance.

IV. PoirIoT Design

This section describes PoirIoT, our in-network system for detecting and identifying IoT devices in high speed networks. PoirIoT can be deployed at edge switches by ISP providers to inspect the communication between a home router and a back-end infrastructure hosted on the cloud[1]. It makes no assumptions about the architecture of the client network, meaning IoT devices can be either behind a NAT or have their IPs directly exposed (more common in IPv6 deployments). We consider device connections as symmetric, i.e., they traverse the same edge switch in both upstream and downstream directions. This

[1]Similarly, it could also be deployed on the gateway of a campus or enterprise network.

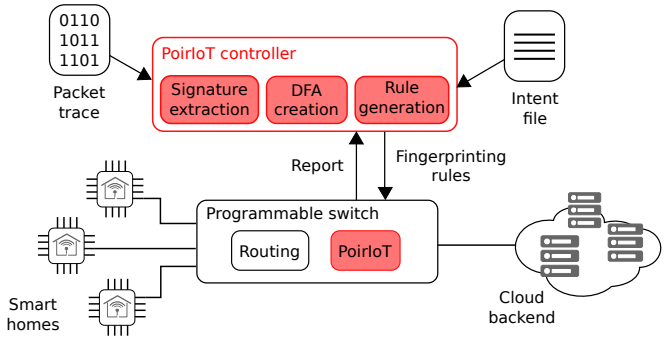

Fig. 1: PoirIoT's architecture. DFA = Deterministic Finite Automata.

is reasonable since ISPs usually connect an edge switch (also called broadband network gateway - BNG) directly to a home router [24][25]. PoirIoT does not rely on payload nor sampled data to work. Figure 1 shows its architecture, which consists of a controller and a programmable switch.

Controller. PoirIoT's controller is primarily responsible for extracting signatures from packet traces (which can be done offline), and converting them into configuration rules for the programmable switch. A signature comprises simple sequences of packet lengths and directions (similar to [11]), which ensures uniqueness for a significant number of devices. This signature design avoids the need for statistical computations (e.g., mean rate or inter-arrival time [6]), which are difficult to implement on current switches. To convert signatures into forwarding rules, PoirIoT's controller first merges them into a deterministic finite automata (optimized to reduce the total number of rules) before installing the generated rules into a finite-state machine (FSM) deployed in the switch data plane. Every signature is also associated with a policy (e.g., raise an alarm to controller, send traffic to an IDS for further processing), which operators can use to implement high-level intents. We explore the deployment and configuration of PoirIoT's FSM in detail in Section IV-C.

Switch. The programmable switch is configured to run PoirIoT's data plane pipeline. Note that the pipeline relies mainly on direction and length information from packets (as part of the device signature), which makes it fully compatible with different protocols (e.g., TCP, UDP). In addition, PoirIoT does not modify any header fields from packets and thus can be modularly coupled with other network functions (e.g., routing, monitoring). Further, all of our processing logic fits within the switch ingress, leaving the egress completely available. We detail PoirIoT's data plane design in Section IV-B. Next, we describe how PoirIoT's controller extracts signatures from packet traces.

A. Signature extraction at PoirIoT's controller

PoirIoT extracts signatures for IoT events (e.g., a device turning on/off) using the same method proposed by Trimananda et al. [11]. This method uses machine learning to identify relevant sequences of packet lengths and directions (i.e., those associated with an event) in a flow. It is based on the observation that these sequences are mostly unique among devices and events on the same device. Similar observations were also made by previous works and have been extensively validated in an empirical fashion in the literature [26][27][16]. We do not claim PoirIoT and its signatures to be applicable for *all* IoT devices/events, but rather to be used complementary to other solutions, e.g., based on traffic shape, volume, or MUD profiles. The main advantage of PoirIoT lies in the fact it is capable of operating efficiently at high traffic rates (i.e., at a Tbps scale), particularly on encrypted traffic.

PoirIoT's signature extraction method takes as input a traffic trace containing packets from/to an IoT device and a set of event timestamps (i.e., annotations). This information could be collaboratively provided by users (e.g., as in [10]) or derived by operators from public datasets[2]. Once the annotated trace is ready, PoirIoT performs a 6-step process to come out with a valid signature. First, it filters out packets that are likely unrelated to the IoT device (**Step 1**). In particular, this step will remove packets whose source or the destination IP addresses do not match that of the IoT device or its controlling smartphone. In addition, PoirIoT also discards packets that do not lie within a given time window (e.g., 15 seconds) after an event, based on the user-provided annotations[3].

Next, the signature extraction method reassembles all flows in the filtered trace (**Step 2**). We identify a flow by its 5-tuple, which contains source and destination IP addresses, source and destination ports, and the transport layer protocol identifier. Similarly to Trimananda *et al.*, we focus on TCP flows in this work, though PoirIoT's signature extraction method can also be applied to other protocols, e.g., UDP, QUIC. In particular, we only allow data packets (i.e., packets carrying actual application data) to become part of a signature for a TCP flow, which increases the chances of this signature to be unique. The intuition is that control packets, e.g., the ones exchanged during the TCP handshake, or TLS key negotiation, likely have the same length among connections and thus could increase the number of mismatches, i.e., false positives or negatives.

Once flows are reassembled, PoirIoT groups consecutive packets in a flow into pairs according to the rules described in Equation 1 (**Step 3**). More specifically, it creates a set of packet pairs where each pair P contains either: a) two consecutive packets if the packets travel in opposite directions; or b) a packet and a special identifier (μ) (e.g., when consecutive packets travel in the same direction or in case a packet is the last one in a flow). We use p_i to represent the i-th packet in a flow.

$$P = \begin{cases} (p_i, p_{i+1}), & \text{if } p_i, p_{i+1} \text{ go in opposite directions} \\ (p_i, \mu), & \text{otherwise} \end{cases} \quad (1)$$

[2]A third alternative is the IoT device manufacturer itself providing the signature, similarly to creating a MUD profile.

[3]We adopt the same time window as Trimananda et al. [11], i.e., 15 seconds.

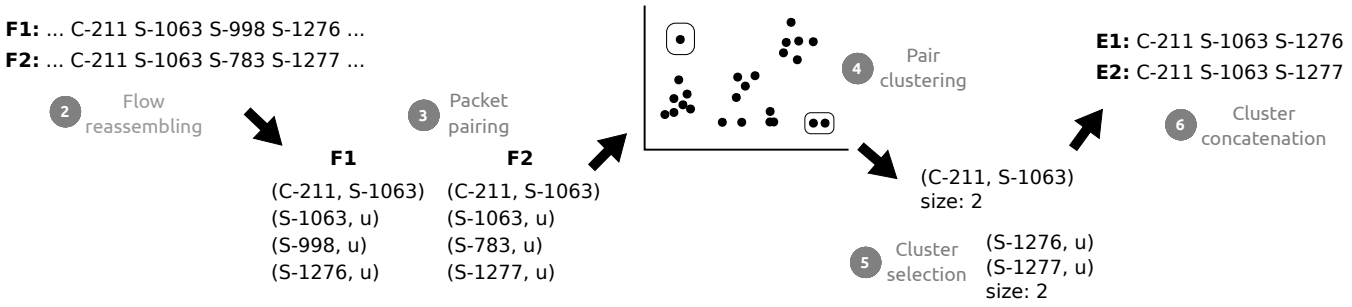

Fig. 2: Example describing PoirIoT's signature extraction method. Trace filtering (i.e., Step 1) was omitted for simplicity. C and S represent packets flowing from client (or IoT device) to cloud and the other way around, respectively.

After pairing packets for all flows in the filtered trace, PoirIoT applies an unsupervised learning algorithm (e.g., DBSCAN [28]) to cluster similar pairs, i.e., pairs whose packet lengths and directions are similar (**Step 4**)[4]. PoirIoT then selects clusters whose size aligns with the number of events reported in the trace to be part of the respective event's signature (**Step 5**). Finally, it concatenates pairs from subsequent clusters (i.e., clusters whose pairs appear subsequently in their respective flows) based on timing information to create the actual signatures (**Step 6**).

Figure 2 shows an example, where two reassembled flows ($F1$ and $F2$) from a filtered trace are processed to generate unique signatures ($E1$ and $E2$) for an event E associated to an IoT device. Note that we omit the trace filtering step (i.e., Step 1) for simplicity. Also, we represent packet directions as either C or S. The former indicates packets travelling from client (or IoT device) to cloud and the latter packets flowing the opposite direction.

B. Data Plane Design

The core of PoirIoT is a packet processing pipeline that deploys an FSM abstraction. Figure 3 shows the pipeline layout as well as the overall packet flow inside the switch. All blocks are deployed as part of the switch ingress pipe. First, PoirIoT checks whether an incoming packet is a re-submission or not. Due to Tofino constraints[5], we use packet re-submissions to update the FSM state (*Set state* block). In case of a new packet, PoirIoT passes it through a length filter which will select only packets whose length matches those of interest (i.e., are part of a device signature) for further processing. In practice, the length filter is implemented as a match-action table that matches packet lengths, e.g., the IPv4 total length field. Unfiltered packets are forwarded as usual.

Whenever PoirIoT selects a packet for further processing (i.e., there is a hit on the length filter), it starts applying its fingerprinting logic. That consists of three main steps. First, it retrieves the current FSM state for the corresponding device (*Get state* block)[6]. Next, PoirIoT identifies the packet direction by matching its incoming port. We consider the existence of access and trunk ports in the switch, i.e., PoirIoT is running in a border network device. Finally, PoirIoT checks whether it should transition its current FSM state (*FSM* block), which occurs when the next signature step is identified, i.e, packet length and direction match. We provide more details on how PoirIoT's FSM works in the next section.

All packets triggering a state transition in PoirIoT's FSM are marked for re-submission (*Set resubmit* block), which happens at the end of the pipeline. In that case, the new state accompanies the re-submitted packet as a metadata. Parallelly, PoirIoT also keeps a timer (indeed multiple *virtual* timers, one for each tracked device) as part of its FSM. Timers are important to ensure the FSM does not get stuck at a wrong state, e.g., due to a false positive transition (see Section VI for more details). Whenever a timeout happens, the respective timer is reset and the packet marked for re-submission to also reset the corresponding FSM (*Set timer/resubmit* block). PoirIoT also resets the timer when a signature matches (i.e., an event/device is detected) or an FSM transitions from its start state, both cases after an FSM hit.

C. Finite State Machine

PoirIoT's data plane implements a finite state machine abstraction for signature matching (FSM block in Figure 3). This process consists of two-steps and is depicted in Figure 4. First, PoirIoT's controller converts a set of signatures into a DFA. As described in Section IV-A, every signature in the input set consists of an unlimited sequence of <direction, packet length> pairs (Figure 4a). Each pair triggers a transition in our DFA leading to a uniquely identified state (Figure 4b). Also, we merge signatures with the same prefix (i.e., initial legs) to reduce the DFA size and consequently the number of generated rules. Ultimately, PoirIoT ends up generating an

[4]Like Trimananda et al. [11], we define similarity as the Euclidean distance between the corresponding packet lengths from two pairs if directions match. Otherwise, distance is maximal.

[5]Tofino restricts access to the same register to be confined into a single pipeline stage, which restricts the operations a program can perform over the register content in a single pass [29].

[6]To optimize resource usage, PoirIoT deploys only a single FSM on hardware and stores its current state individually for each tracked device. This creates the abstraction of multiple *virtual* FSMs (one for each device) in the switch. The current state is then retrieved per device by hashing relevant traffic keys, e.g., source IP or IP/port pair in case the device is behind a NAT.

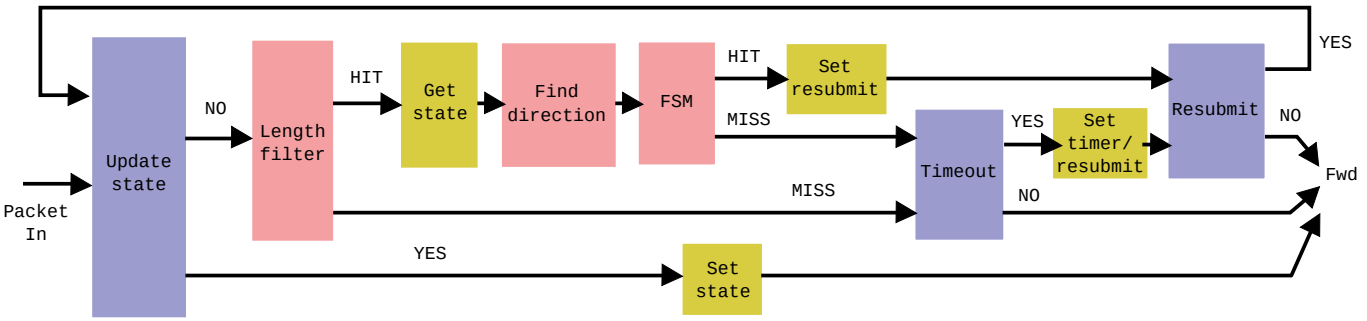

Fig. 3: PoirIoT's data plane layout.

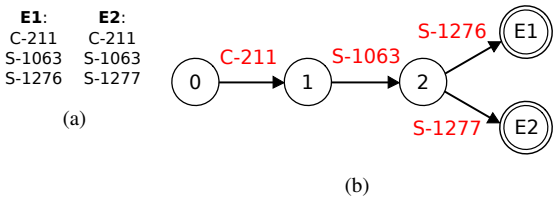

Fig. 4: DFA for signatures E1 and E2.

Match			Action
State	Dir	Length	
0	C	211	set_state(1)
1	S	1063	set_state(2)
2	S	1276	report_event(E1)
2	S	1277	report_event(E2)

Fig. 5: FSM transition table with entries for signatures E1 and E2.

acyclic DFA, i.e., a tree, whose final states point to user-defined policies like reporting the event or redirecting the associated traffic towards an IDS.

Once the DFA is created, it is compactly implemented using a state transition table that maps a current state, a direction and a packet length to a next state (or a policy action in case of an end state), as shown in Figure 5. The new state is then stored in the register associated with the tracked device/event for further processing of the candidate signature (*Set state* block from Figure 3). As each packet can trigger at most one state transition in PoirIoT's FSM, it is not worth replicating the latter to speed up the fingerprint process. Moreover, the size of the state transition table increases linearly with the number of signatures in the worst case, meaning a single switch can support a large number of them (see Section V-E for more details). In practice, we found the state transition table to contain no more than a few hundred entries for the scenarios we tested.

V. Evaluation

We implement PoirIoT data plane in P4 (around 500 lines of code) and compile it to Intel® Tofino™ ASIC [8]. Our code is available at [9]. The control plane runs in Python (around 300 lines of code) and translates device signatures into commands for the switch runtime API. We use the method outlined in Section IV-A to extract signatures directly from traffic traces. In this section, we provide evaluation results on PoirIoT. The results demonstrate that PoirIoT can accurately detect a significant number of IoT devices (§V-B), provides noticeable performance improvements compared to a server-based implementation (§V-C), and incurs a small resource overhead on the programmable switch ASIC (§V-D).

A. Setup

Our testbed consists of a Wedge 100BF-32X 32-port programmable switch with a 3.2Tbps Tofino ASIC and two servers each equipped with an Intel® Core™ i7-9700 CPU @ 3.00GHz, 16GB RAM, and a 40Gbps Agilio® LX SmartNIC. The servers run Ubuntu 18.04 with kernel version 4.15, and both are directly connected to the switch. We use one server as a traffic generator while the other runs a state-of-the-art IoT device fingerprinting system, PingPong, for comparison purposes.

Our workload is comprised of a real-world dataset [11] containing packet traces from 19 popular smart home devices (e.g., smart plugs, light bulbs, thermostats, home security systems) from 16 different vendors. We use *tcpreplay* version 4.2.6 to replay these traces in our traffic generator, which starts multiple threads to achieve rates above 1 Gbps. Finally, we configure PoirIoT with signatures for 14 devices extracted from the same dataset. For that, we discarded devices whose traffic involves only phone-device communication and thus cannot be seen by an ISP (see Section VI for more details on signature types). Note that our results report PoirIoT's performance with respect to the set of detectable devices.

B. Micro-benchmarking

Device detection. First, we evaluate the performance of PoirIoT for detecting different IoT devices. We consider a device as detected if any of its signatures (which represent

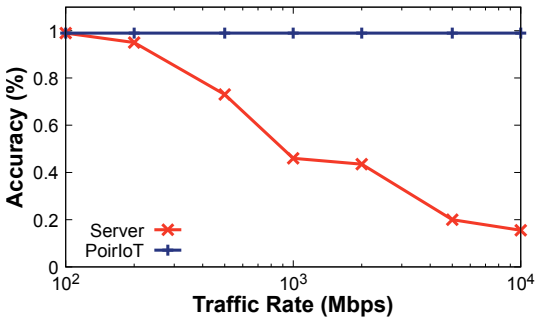

Fig. 6: Amazon plug events detected.

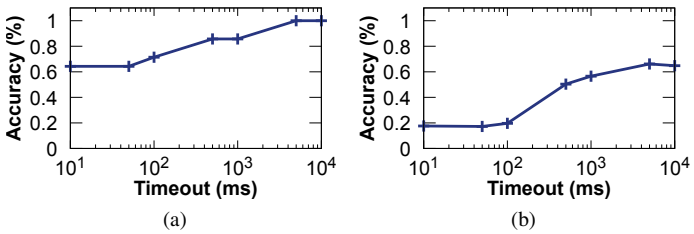

Fig. 7: PoirIoT's accuracy for detecting different IoT devices (a) and their particular events (b).

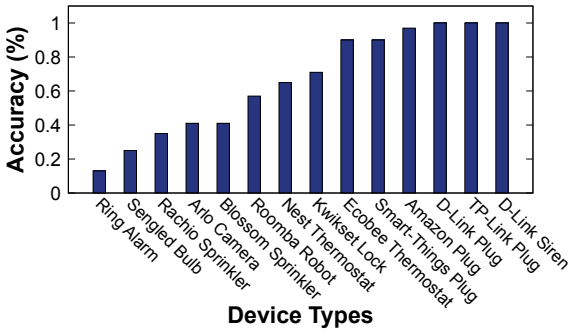

Fig. 8: Events detected per device using a 5s timeout.

different events) is matched. Figure 7a shows PoirIoT's accuracy (or the ratio of devices detected) as we vary its timeout interval, which is used to reset the FSM state. We can see that the accuracy improves as we increase the timeout value, reaching 100% (i.e., *all* devices detected) around 5 seconds. Overall, longer timeouts increase the chances all packets in a signature will show up before the FSM resets. Timeouts that are too long, however, may cause the FSM to get stuck in a wrong state, and consequently miss packets that are part of a devices' signature. For example, PoirIoT's device detection accuracy drops to around 80% when there are no timeouts at all, though this is a worst case scenario.

Event detection. Next, we look at how PoirIoT performs with respect to event detection, e.g., a device is on/off. The ability to identify specific events, and ultimately the device's behavior, can be used by ISPs to detect misbehaving devices and quickly react to malicious activities. For example, an ISP could generate signatures for compromised devices and use them to detect attempts to exfiltrate data or contact a command and control server in a botnet [30], all at line rate and with no need to inspect the packet's payload.

Figure 7b shows PoirIoT's event detection accuracy as we vary its timeout. Similar to device detection, longer timeouts (above 500 ms for the evaluated dataset) tend to produce better results, while overestimated ones may lead to blocking the FSM and consequently missing relevant events. Interestingly, PoirIoT's event detection accuracy peaked at 71% for the evaluated dataset, meaning that even for the same device some events may be detected while others may not once we set a timeout. To better understand this effect, we investigate PoirIoT's event detection accuracy for each device in the evaluated dataset.

Figure 8 shows the results for a timeout of 5 seconds. As we can see, there is a large variation in the detection accuracy, ranging from 13% in the worst case (Ring alarm) to 100% in the other extreme (e.g., D-Link plug). This is mainly due to the fact that different events (e.g., switching on and off) or even the same event (e.g., a motion event) can have different duration on the same device, so a single timeout value may not be the most appropriate choice. We discuss further timeout policies in Section VI.

C. Performance improvement

To assess the benefits of moving IoT device fingerprinting to a programmable switch, we compare PoirIoT with PingPong [11], a state-of-the-art server-based device fingerprinting system. For this experiment, we send a stream of packets containing a single device capture at varying speeds and report the ratio of detected events for both PoirIoT and PingPong (server). We use a single device capture (rather than an all device scenario) to isolate the effect of an increasing load on the detection accuracy and hence ensure a fair comparison. Note that we report results for Amazon Plug as its event detection accuracy is close to 100% for both systems (i.e., server and switch-based) under low loads, which enables us to easily observe performance variations. *All* devices we tested (fourteen in total) have shown a similar trend. Figure 6 presents the results. As we can see, PingPong's accuracy drops significantly, down to 15.5%, as the rate increases. This reduction is mainly because packets start getting dropped once the rate exceeds the server's buffering and processing capacity, which compromises the signature matching process. The switch, on the other hand, can operate at line rate and thus is able to maintain its accuracy even under high loads.

D. Resource usage

Next, we evaluate how much ASIC resources PoirIoT consumes based on the P4 compiler's output. Table I shows the results. Overall, our system consumes less than 5% of the ASIC resources leaving ample room for other network functions (e.g., routing, load balancing). Memory is the most occupied resource as PoirIoT needs to store both state and

TABLE I: Resource utilization.

Resource	Usage
Match Crossbar	3.1%
SRAM	5.1%
TCAM	0%
VLIW Instruction	3.4%
Hash Bits	4.7%

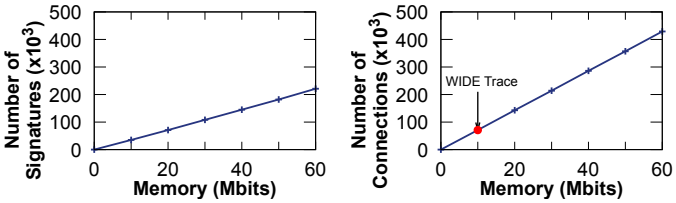

Fig. 9: Number of (a) signatures and (b) connections PoirIoT can support for a given amount of memory. Results depict the worst case scenario (i.e., a uniform distribution of packet sizes).

timing information for every device it is trying to detect. In particular, the 5.1% SRAM usage can support the detection of up to 65K simultaneous devices (an average of 4 devices per household on a typical ISP network [31]). This value can be tweaked based on the network size though, as the switch has capacity to track hundreds of thousands of devices (see Figure 9b and the discussion in the next section).

E. Scalability

Finally, we analyze how our system performs at a large scale. For that, we generate a synthetic set of 4-way signatures (i.e., signatures of length four) as an extension to our original signature set, and test the resulting FSM on the same dataset described in Section V-A. Packet sizes from synthetic signatures follow either a uniform or a derived distribution, the latter mimicking the distribution of our original signature set. Note that the uniform distribution is useful to assess a worst case scenario with respect to both resource consumption and wrong state transitions (i.e., false positives and negatives) as it maximizes the chances of erroneously matching a packet. In addition to an increased number of signatures, we also play with a larger (non IoT-based) dataset to analyze how the background traffic impacts PoirIoT's performance. More specifically, we replay a whole day of traffic (September 11, 2021) captured from a transit link as part of the WIDE project [32]. This trace contains more than 35M packets spread over approximately 70K connections.

Signature set. Increasing the number of signatures in PoirIoT's FSM demands additional resources from the underlying switch ASIC, particularly memory. In this sense, Figure 9a shows the number of signatures PoirIoT can support as we vary the amount of available memory. For simplicity, we only show results for the worst case scenario (i.e., signatures generated based on a uniform distribution of packet sizes). In this scenario, chances of merging two FSM states are minimal and consequently the amount of generated rules is maximal. As we can see, PoirIoT has ample room for device signatures, supporting more than 35K on a single pipeline stage (i.e., 10 Mbits of memory). In practice, providers may not need more than a few hundred signatures to track the most relevant devices (e.g., the ones with known critical vulnerabilities or frequently involved in DDoS attacks) depending on their policy [3].

Tracked connections. In addition to signatures, PoirIoT also consumes significant amount of ASIC resources for connection tracking. More specifically, our system fingerprints each connection independently and requires a register per connection to store its state and corresponding timing information. The latter is necessary to implement PoirIoT's timeout mechanism. Also, PoirIoT uses connection information at different stages in the packet processing pipeline, meaning the same information must be stored at multiple points (twice in our case) due to constraints on how Tofino manipulates state [29].

Figure 9b shows the number of connections PoirIoT can track as we vary the available memory. Note that the required memory splits over two pipeline stages in our design. We can observe that PoirIoT is able to track more than 400K connections, largely above current demand on average-sized ISPs [31]. To put this result in perspective, we also plot the memory requirement for tracking relevant IP connections (i.e., connections where at least one packet hits an entry in PoirIoT's length filter) in a whole day of traffic from the WIDE trace. As we can see, the demand is around 10 Mbits which fits well into the switch memory capacity. Interestingly, PoirIoT requires more memory to store a signature than a connection state, which stems from the fact that a single signature can span over multiple forwarding rules in PoirIoT's FSM.

Detection accuracy. We also assess the effect of a growing number of signatures on PoirIoT's accuracy. Intuitively, the larger the number of signatures the higher the chances of making wrong transitions and consequently missing an IoT device or event. Figure 10 shows PoirIoT's device detection accuracy as we vary its number of signatures. We can see that PoirIoT is able to sustain high accuracy (i.e., close to 100%) up to 250 signatures. Beyond that, accuracy starts gradually degrading due to the higher chances of false positive transitions. To put our scale in perspective, the Mirai botnet has around ninety identified IoT devices [3], and the dashed green line in the figure shows the number of signatures derived from our working IoT dataset (see Section V-A). We can also observe that PoirIoT's accuracy degradation is more prominent for signatures following a uniform packet size distribution (up to 15% worse at 550 signatures), meaning it can perform reasonably better at a large scale if the signature set is tuned to a particular group of devices (e.g., the most common ones in a given botnet).

Figure 11 depicts PoirIoT's event detection accuracy as we increase the number of signatures. Similarly to device detection, accuracy tends to be better (up to 20% higher) for

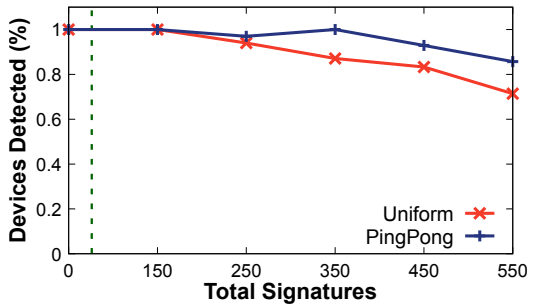

Fig. 10: PoirIoT's device detection accuracy for an increasing number of generated signatures. Dashed line indicates the number of real signatures extracted from PingPong dataset.

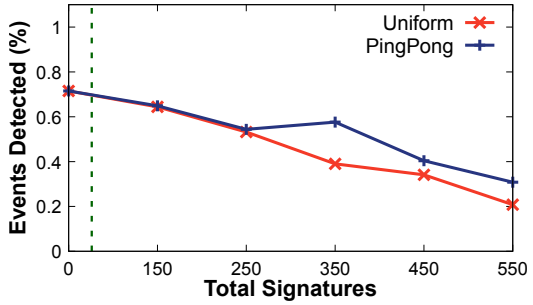

Fig. 11: PoirIoT's event detection accuracy for an increasing number of generated signatures. Dashed line indicates the number of real signatures extracted from PingPong dataset.

targeted, i.e., non-uniform, signatures. However, event detection performance is considerably worse at scale, dropping to an around 20% accuracy in the worst case. That stems mainly from the fact event detection requires fine-grained packet identification (i.e., identifying all packets from an event) to succeed while it suffices identifying a single event to detect an IoT device. Despite the low accuracy, it is possible to augment PoirIoT's event detection capabilities by, e.g., splitting its FSM into multiple smaller ones. We leave investigating that as future work.

VI. Discussion

Flexible timeout policies. PoirIoT's timeout mechanism may be negatively impacted when: i) IoT devices have significantly different communication patterns (e.g., in terms of request-reply intervals); or ii) network delays due to congestion or quality of service queuing are at the same timescale as device activities. While longer timeouts have shown to perform better in our evaluated scenarios, there is a turning point where accuracy starts decreasing as the whole system may get stuck at a wrong state (e.g., due to a false positive packet). An alternative would be implementing more flexible timeout policies (e.g., based on exponential backoff), but that requires re-engineering our data plane design. Moreover, more complex policies tend to also require more resources which may not always be feasible in a programmable switch ASIC. We leave investigating the implementation and evaluation of such policies in the switch data plane as future work.

Signature extraction. Usually, there are three main types of traffic exchanges in an IoT application: device–phone, device–cloud, and phone–cloud communication [11]. In theory, a signature could contain packets of any of these types, though in practice that is not always possible. First, device–phone traffic can frequently only be seen at a local area network, making it more difficult for an ISP to (having the client's consent) run a monitoring solution at every single home. Second, phones are actually mobile appliances that may not always be hosted on the same network as its paired IoT device. That also makes signatures based on phone–cloud communication less reliable, though they would still work when a client is at home. As part of our future work, we plan to investigate better algorithms for extracting signatures from a bigger set of IoT devices.

Ethical and privacy considerations. The ability to fingerprint IoT devices in the wild may raise ethical/privacy concerns. However, the main goal of this paper is not to enable ISP/network providers to inspect client information without their consent. Instead, deployments should be in compliance with applicable laws (e.g., GPDR [33]) and only collect data from clients that have explicitly agreed to share it. We envision PoirIoT to be used in a constructive manner, to help both providers and clients improve their network security capabilities (e.g., by identifying and patching vulnerable IoT devices). Ideally, clients would need to enroll in an *opt-in* process to have their homes monitored by ISPs. If that is not the case (e.g., when ISPs are adversarial), an alternative is to deploy obfuscation mechanisms such as traffic padding [15] or blocking [34] to prevent device fingerprinting.

VII. Conclusion

As the popularity of household IoT devices continues to grow, so to does the potential for misuse and security breaches. To mitigate the threat presented by compromised IoT devices, the solutions must expand as well. As part of this expansion, we present PoirIoT an in-network system for fingerprinting IoT devices. This system uses signatures derived from device-cloud exchanges to uniquely identify a device. By utilizing the capabilities of programmable switches, our system can inspect every packet that passes through it, enabling high granularity detection at rapid speeds. This work is the first step in realizing a complete, detection to intervention, in-network system for protecting against malicious IoT traffic.

Acknowledgements. We thank Pulkit Garg for his contributions on early stages of this work. This research is supported by NSERC CGS-M and Discovery Grants.

References

[1] S. Stasha, "An in-depth view into smart home statistics," 2021. [Online]. Available: https://policyadvice.net/insurance/insights/smart-home-statistics/

[2] K. L. Lueth, "Iot 2020 in review: The 10 most relevant iot developments of the year," *IoTAnalytics*, Jan 2021. [Online]. Available: https://iot-analytics.com/iot-2020-in-review/

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *Proceedings of the 26th USENIX Conference on Security Symposium*, ser. SEC'17. USA: USENIX Association, 2017, p. 1093–1110.

[4] G. Chu, N. Apthorpe, and N. Feamster, "Security and privacy analyses of internet of things children's toys," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 978–985, 2019.

[5] S. J. Saidi, A. M. Mandalari, R. Kolcun, H. Haddadi, D. J. Dubois, D. Choffnes, G. Smaragdakis, and A. Feldmann, "A haystack full of needles: Scalable detection of iot devices in the wild," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 87–100.

[6] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '20, 2020, p. 207–218.

[7] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, Jul. 2014.

[8] Intel, "Intel tofino," 2021. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html

[9] P. Authors, "Poiriot implementation," 2021. [Online]. Available: https://github.com/PINetDalhousie/poiriot

[10] D. Y. Huang, N. Apthorpe, F. Li, G. Acar, and N. Feamster, "Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 4, no. 2, Jun. 2020.

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

[12] S. Leibson, "Aps networks launches three openbng broadband network gateways incorporating intel® xeon® d processors, intel® tofino™ switch asics, and intel® stratix® 10 mx fpgas," *Intel Programmable Logic*, Jun 2021. [Online]. Available: https://blogs.intel.com/psg/aps-networks-launches-three-openbng-broadband-network-\ gateways-incorporating-intel-xeon-d-processors-intel-tofino-switch\ -asics-and-intel-stratix-10-mx-fpgas/

[13] Y. Zhao, K. Yang, Z. Liu, T. Yang, L. Chen, S. Liu, N. Zheng, R. Wang, H. Wu, Y. Wang, and N. Zhang, "Lightguardian: A full-visibility, lightweight, in-band telemetry system using sketchlets," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, Apr. 2021, pp. 991–1010.

[14] H. Guo and J. Heidemann, "Detecting iot devices in the internet," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2323–2336, 2020.

[15] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster, "Keeping the smart home private with smart(er) iot traffic shaping," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 128–148, 2019. [Online]. Available: https://doi.org/10.2478/popets-2019-0040

[16] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, "Homesnitch: Behavior transparency and control for smart home iot devices," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '19, 2019, p. 128–138.

[17] A. Bremler-Barr, H. Levy, and Z. Yakhini, "Iot or not: Identifying iot devices in a short time scale," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9.

[18] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: An analysis of

[20] C. Duan, S. Zhang, J. Yang, Z. Wang, Y. Yang, and J. Li, "Pinball: Universal and robust signature extraction for smart home devices," iot devices on home networks," in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA, Aug. 2019, pp. 1169–1185.

[19] Y. Afek, A. Bremler-Barr, D. Hay, R. Goldschmidt, L. Shafir, G. Avraham, and A. Shalev, "Nfv-based iot security for home networks using mud," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9.
in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021, pp. 1–9.

[21] S. Bai, H. Kim, and J. Rexford, "Passive os fingerprinting on commodity switches," 2019. [Online]. Available: https://www.cs.princeton.edu/∼jrex/papers/p40f.pdf

[22] J. Kim, H. Kim, and J. Rexford, "Analyzing traffic by domain name in the data plane," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '21. New York, NY, USA: Association for Computing Machinery, 2021.

[23] T. Jepsen, D. Alvarez, N. Foster, C. Kim, J. Lee, M. Moshref, and R. Soulé, "Fast string searching on pisa," in *Proceedings of the 2019 ACM Symposium on SDN Research*, ser. SOSR '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 21–28.

[24] R. Kundel, L. Nobach, J. Blendin, H.-J. Kolbe, G. Schyguda, V. Gurevich, B. Koldehofe, and R. Steinmetz, "P4-bng: Central office network functions on programmable packet pipelines," in *2019 15th International Conference on Network and Service Management (CNSM)*, 2019, pp. 1–9.

[25] J. Hu, Z. Zhou, X. Yang, J. Malone, and J. W. Williams, "Cablemon: Improving the reliability of cable broadband networks via proactive network maintenance," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 619–632.

[26] H. Gordon, C. Batula, B. Tushir, B. Dezfouli, and Y. Liu, "Securing smart homes via software-defined networking and low-cost traffic classification," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2021, pp. 1049–1057.

[27] A. J. Pinheiro, J. de M. Bezerra, C. A. Burgardt, and D. R. Campelo, "Identifying iot devices and events based on packet length from encrypted traffic," *Computer Communications*, vol. 144, pp. 8–17, 2019.

[28] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[29] N. Gebara, A. Lerner, M. Yang, M. Yu, P. Costa, and M. Ghobadi, "Challenging the stateless quo of programmable switches," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, ser. HotNets '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 153–159. [Online]. Available: https://doi.org/10.1145/3422604.3425928

[30] O. Alrawi, C. Lever, K. Valakuzhy, R. Court, K. Snow, F. Monrose, and M. Antonakakis, "The circle of life: A large-scale study of the iot malware lifecycle," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021.

[31] M. Trevisan, D. Giordano, I. Drago, M. M. Munafò, and M. Mellia, "Five years at the edge: Watching internet from the isp network," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 561–574, 2020.

[32] K. Cho, K. Mitsuya, and A. Kato, "Traffic data repository at the wide project," in *USENIX 2000 FREENIX Track*, San Diego, CA, June 2000.

[33] Council of European Union, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)," *OJ*, vol. L 119, pp. 1–88, 2016-05-04.

[34] A. M. Mandalari, D. J. Dubois, R. Kolcun, M. T. Paracha, H. Haddadi, and D. R. Choffnes, "Blocking without breaking: Identification and mitigation of non-essential iot traffic," *CoRR*, vol. abs/2105.05162, 2021. [Online]. Available: https://arxiv.org/abs/2105.05162