The Forest Behind the Tree: Revealing Hidden Smart Home Communication Patterns

François De Keersmaeker
UCLouvain
Louvain-la-Neuve, Belgium
francois.dekeersmaeker@uclouvain.be

Rémi Van Boxem
Junia
Lille, France
remi.van-boxem@student.junia.com

Cristel Pelsser, Ramin Sadre
UCLouvain
Louvain-la-Neuve, Belgium
firstname.lastname@uclouvain.be

***Abstract*—The widespread use of Smart Home devices has attracted significant research interest in understanding their behavior within home networks. Unlike general-purpose computers, these devices exhibit relatively simple and predictable network activity patterns. However, previous studies have primarily focused on normal network conditions, overlooking potential hidden patterns that emerge under challenging conditions. Discovering the latter is crucial for assessing device robustness.**

This paper addresses this gap by presenting a framework that systematically and automatically reveals these hidden communication patterns. By actively disturbing communication and blocking observed traffic, the framework generates comprehensive profiles structured as behavior trees, uncovering traffic flows that are missed by more shallow methods. This approach was applied to ten real-world devices, identifying 254 unique flows, with over 27% only discovered through this new method. These insights enhance our understanding of device robustness, and the thus obtained profiles provide a more complete description of the network behavior of devices, as needed, for example, for the configuration of security solutions.

***Index Terms*—IoT, Smart Home, networks, robustness, security, traffic profiling**

I. Introduction

In the **I**nternet **o**f **T**hings (IoT) paradigm, physical objects are equipped with sensing, computing, and networking capabilities. This allows them to monitor or react to their environment and exchange messages with other objects and with remote servers [1]. A common use case of this paradigm is the Smart Home, composed of household objects, ranging from small power outlets to large appliances. The popularity of Smart Home devices has increased sharply in the past years, with a market size estimated at 101.07 billion USD in 2024 [2]. Their main appeal is to provide home automation, and therefore convenience, to the user.

As IoT devices have very low computing resources, it is a technical challenge to embark them with all the technology necessary for their correct operation. Notoriously, to stay economically competitive, manufacturers tend to expand the user-visible functionalities on the devices' limited hardware, neglecting transversal aspects such as robustness or security [3], [4]. Ironically, this compromise might impede the initial incentive behind the design of Smart Home systems, i.e. user convenience. Indeed, if devices do not provide robustness in the face of network communication instability, they may become unresponsive, disrupting the whole system.

979-8-3315-0376-5/25/$31.00 ©2025 IEEE

The motivation behind this work is to assess the robustness of Smart Home devices against network instabilities. More precisely, we aim to discover and describe network communication patterns issued by such devices in the case where their default traffic fails.

Smart Home devices usually exhibit simple and predictable network communication patterns [5] under ideal conditions. Consequently, researchers have designed solutions to express their network behavior in a compact form, a so-called *profile*. The IETF has even standardized a format for such profiles, the **M**anufacturer **U**sage **D**escription (MUD) [6], albeit admittedly relatively limited [7]–[9]. Several methods have been presented in the past to automatically create the profile of a device [10]–[12]. For this purpose, its network communications are observed over a certain period of time, in the wild or under laboratory conditions. Typically, relatively complex experimental setups are required [13], [14], due to the variety of possible interactions with the device. Data mining and analysis techniques are then used to extract compact representations of the network traffic and associate them with the device, or even with specific device events.

We argue that existing approaches to profiling are insufficient for our goal of gaining comprehensive insight into the network behavior of Smart Home devices, as they do not aim at triggering communication patterns that only become visible under certain network conditions, in particular when the default communication mechanism does not succeed. For example, an IoT device might have a list of alternative domain names for its cloud hosted services that it will only contact if the default domain name fails to resolve or the server behind that name does not respond; or it can switch from using TCP to UDP (and vice-versa), if one of the protocols is blocked. Ignoring such behaviors leads to profiles, in which some of the device's communication patterns are missing or incompletely described, making it difficult to grasp if a device is robust with regard to network events.

In this paper, we present a framework, mostly automated and requiring little prior configuration, to comprehensively uncover the communication patterns of a Smart Home device. For each type of interaction with the device, it first observes the traffic flows that occur in an unconstrained network and

builds a set of flow descriptors from them. Such descriptors contain Internet and transport-layer information, as well as selected, available application-layer data, such as domain names. The system then iteratively blocks flows corresponding to certain descriptors and repeats the interaction with the intent of making new flows appear. The algorithm selecting which flows to block and the sequence of blocking experiments is not straightforward. Our aim is to discover as many hidden flows as possible but keep the number of experiments under control. To do this efficiently and not end up in an infinite loop where the same patterns appear over and over again, we store the found descriptors in a tree that we traverse using a breadth-first strategy, and prune using an *ad hoc* heuristic. The result is a set of *multi-level*, tree-shaped profiles (one for each type of interaction) for the device that describe the device's default and alternative communication patterns that we then explore to assess the robustness of the device's network communications.

Conceptually, the profiles obtained by our approach can be treated as enhanced versions of the aforementioned ones, such as MUD, as they encompass the base communication patterns, and augment them with the identified hidden patterns. Therefore, they can also be used as input configuration for allow-list firewalls. Current state-of-the-art approaches, which do not cover alternative communication patterns, might consider the latter as malicious, whereas they are actually part of the device's intended behavior, only rarer. Profiles obtained from our framework can thus provide more accurate security configurations that do not block the legitimate communications of a robust device.

The contributions of our work can be summarized as follows:

- **We develop and implement a framework**, modular and easily extensible, which instruments Smart Home devices and **extracts multi-level profiles** for them, in a **mostly automated way**. The core of the framework is a new algorithm that, for a given device interaction, builds an event signature, i.e., the description of flows appearing for that interaction, and then iteratively blocks previously observed flows, until the interaction fails or no new flow is discovered.
- **We apply our framework** to a testbed network comprising ten off-the-shelf devices, and generate multi-level profiles for 36 unique events, which sum up to a total of 254 unique network flows discovered, of which 70 were "hidden", i.e., not part of the default communication patterns of the tested devices, accounting for over 27% of the total number of unique flows, representing the portion that traditional techniques are likely to miss.
- **We assess the robustness** of the instrumented devices, by computing robustness-related metrics. We conclude that most devices provide at least one backup communication strategy, if a default communication pattern fails.
- **We publish the source code** of our signature extraction algorithm at https://github.com/smart-home-network-security/signature-extraction, and of our experimental framework, **including the captures of the testbed's traffic** at https://github.com/smart-home-network-security/smart-home-hidden-communication-patterns.

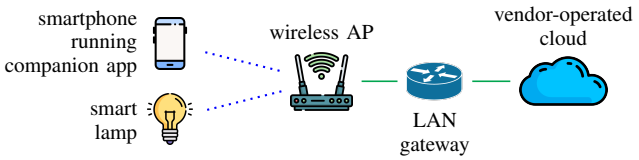

Fig. 1: Typical Smart Home network with a smart lamp

The rest of the paper is structured as follows. We describe the problem and the scope of our work in Section II. In Section III, our methodology is explained in detail. Its evaluation is presented in Section IV. Section V presents related work and positions them compared to ours. Ultimately, the paper concludes in Section VI.

II. Problem Statement and Scope

Fig. 1 shows a typical Smart Home setup with a smart lamp that can be controlled by a smartphone application (called *companion app* in the following). The possible network activities in this setup are highly device specific. The device might communicate directly with the companion app in the local home network, or do this via a manufacturer or vendor-operated cloud server (the latter making it also possible to control the device from outside the home network and to receive firmware updates). In fact, many of the devices on the market have both local and remote communication capabilities.

In line with existing work on profiling discussed in Section V, we assume that the network activity is triggered by an interaction with the Smart Home device. The aim of our work is to obtain a profile, i.e., a description of the network communication patterns associated to that device and interaction. We argue that existing approaches on device profiling, that mostly consist in performing the interaction and recording the resulting network activity, are not sufficient to observe all possible communications patterns. Indeed, we expect that some of them only appear when the default (or configured) patterns fail or are disturbed. Our goal is to discover such patterns in an automated way.

The solution that we will present in the next sections requires the collection, inspection and filtering of packets sent to and received from the Smart Home device and the device running the companion app. If the traffic is unencrypted, we leverage its payload. We do not analyze or modify the device firmware, the companion app, or the server side software. We also do not try to decrypt network traffic, therefore we do not consider information in encrypted payload. However, numerous research has shown that relying on unencrypted information and packet or flow metadata, can already provide sufficient information to accurately fingerprint IoT devices [11], [12], [15].

The terms IoT and Smart Home are quite fuzzily defined in the literature. Our work focuses on domestic devices,

augmented with IoT capabilities, which exhibit a precise, goal-oriented, and limited set of functions, e.g. power plugs, light bulbs, and cameras. More powerful/general-purpose smart home devices (e.g. smart TVs or speaker hubs) do not exhibit such a precise set, and are therefore considered out of scope. Speakers usually act as an intermediary between the user and the end device, enabling the user with voice control. Additionally, such devices allow installing third-party apps which provide them with limitless capabilities. This makes the network behavior of such devices very complex and customizable, rendering an automated analysis with our framework practically unfeasible. Indeed, as the set of functions is unlimited and not known *a priori*, a single device can showcase an unbounded amount of different tree profiles, as one profile pertains to one function.

Similarly to related research works, we only consider IP traffic, either wired or wireless. Other protocols exist for Smart Home devices, such as Zigbee [16] and Thread [17] over IEEE 802.15.4 [18], or the proprietary Z-Wave [19]. Such protocols usually rely on a gateway or hub to connect to the local home network. For devices using these protocols, we profile the gateway's IP traffic. Devices that rely on long-range communication technologies, such as 5G or LoRaWAN [20], are not in the scope of this paper since their traffic cannot be locally filtered or blocked.

Finally, it should be mentioned that devices also communicate for reasons other than being triggered by interactions. Examples include periodic heartbeat messages or messages to discover the local network [14], [21]. A possible approach to profile such activities is to passively monitor the device over a longer period and extract the traffic using periodicity models [14]. Our approach does not pursue such communication activities further, as in this paper we focus on the more complex case of interaction-triggered communication patterns.

III. Profiling Methodology

In this section, we describe our profiling methodology. We start with an outline of our approach, followed by a detailed presentation of its components.

A. General workflow

Fig. 2 gives an overview on the workflow of our methodology to obtain a profile for a Smart Home device under a certain type of interaction. An interaction can require the usage of a triggering source, such as a companion app installed on a smartphone. In practice, a Smart Home device may support different interactions (e.g., for a smart lamp: switching on/off, changing its color, etc.) and triggering sources, and the workflow described here must be repeated for each of them. The workflow consists of multiple steps summarized below. Besides initial manual configuration in Step 1, the rest of the workflow is automated.

Step 1 consists in performing the interaction with the device under investigation and recording the network traffic. To this end, we connect the Smart Home device (or, for non-IP devices, their gateway or hub) and the device running the

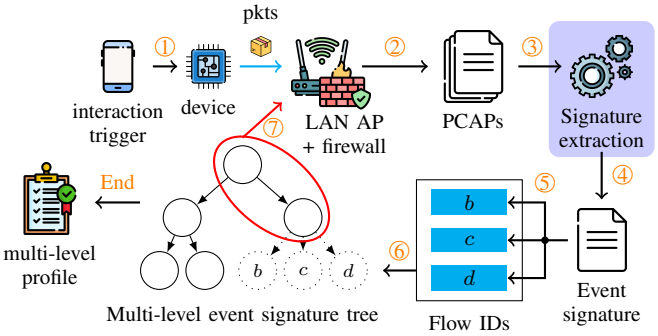

Fig. 2: Overview of the profiling workflow

triggering source to a wireless (resp. wired) LAN and capture the traffic on the Access Point (AP) (resp. switch). We call an interaction and the resulting device state an *event*.

In step 2, the traffic traces obtained are filtered and analyzed, and an *event signature* is extracted. The latter consists of a set of traffic flow descriptors, called *Flow IDs* in the following, that describe the relevant features of the bidirectional flows observed in the traces, and therefore associated with the event. A Flow ID contains network and transport layer information, such as host addresses and port numbers, as well as application layer information (when available). The extracted Flow IDs are stored in a tree-shaped data structure that allows tracking which Flow IDs have already been observed.

In step 3, we select a Flow ID from the tree, configure a firewall running on the switch or AP to block packets matching that ID, and repeat steps 1 and 2. The idea is that by deliberately blocking some of the traffic associated with the interaction and repeating the steps 1 to 2, the device and/or app will try alternative communication patterns.

The steps above are repeated until no new patterns are discovered. The final result is the profile of the device for the investigated event. It takes the form of a tree, where each node is a Flow ID linked to the event, and the path from the root towards a given node gives the set of parent Flow IDs which were blocked to let the current Flow ID appear. We define the Flow IDs initially occurring without any traffic blocking required as *first-level* Flow IDs, whereas the ones appearing as a result of traffic blocking are *hidden* Flow IDs. The latter are the communication patterns which can be discovered only through our multi-level approach, and therefore missed by state-of-the-art techniques.

B. Step 1: Traffic capturing

Similarly to *PingPong* [11] and *IoTAthena* [15], we start capturing the network traffic, perform the interaction, then stop the traffic capture after a predefined duration d. We repeat each interaction and the traffic capturing m times. Our framework automates interactions triggered by a companion app by leveraging the Android Debug Bridge (ADB) tool [22] to generate touch events on the smartphone running the companion app. The only manual configuration required by the framework is to plug in the device under test and to connect it

to the LAN, as well as setting up the framework with the smartphone's screen coordinates to issue the touch events. We also consider power-cycling the Smart home device as an interaction (called *boot interaction* in the following). To automate the boot interaction, we plug the device in a smart power outlet controlled by our framework.

We collect all incoming and outgoing traffic of the Smart Home device and of the smartphone, since we consider the activities of both of them potentially relevant for the characterization of the event. Between captures, we introduce random waiting intervals to avoid the systematic capturing of periodic background traffic unrelated to the interaction. Control plane packets are filtered out, in particular all ARP, ICMP, DHCP packets, TCP Handshake packets, and TLS Handshake packets, except `Client Hello` packets containing the Server Name Indication (SNI) extension.

As our approach is based on blocking traffic, it can happen that the interaction does not cause the desired state change in the Smart Home device. For the event signature extraction described in step 2, it is important to filter out such unsuccessful event executions. We do this by verifying the device state, e.g. whether the lamp has turned on, after each capture. To obtain the device state, we leverage third-party libraries designed to control the device, e.g. `python-kasa` [23] for the TP-Link HS110 plug, while ensuring this does not interfere with the traffic related to the studied interaction. We believe this method is more precise and reliable than the approach followed by Mandalari *et al.* [24] which compares screenshots of the companion app to check whether the displayed status of the Smart Home device has changed. If we did not find such a library, or we failed to include it in our framework, we fall back to screenshots, too, but we compute the Structural Similarity Measure (SSIM) [25] and consider two screenshots identical if the SSIM is over an empirically defined threshold, instead of directly comparing image pixels like Mandalari *et al.* The result are $m^+ \leq m$ traffic captures of successful event executions.

C. Step 2: Event signature extraction

For each of the m^+ packet traces obtained in step 1, the recorded packets are first aggregated to Flow IDs, and then the event signature is extracted. This happens in multiple steps that we now describe.

1) Replacing IP addresses by domain names: During an event, the Smart Home device or the companion app may communicate with cloud services provided by the device vendor or other external hosts. Where possible, we replace non-local source and destination IP addresses by domain names. This makes the signature extraction more robust against changes in the IP addresses caused by cloud migrations or load balancing. To do so, we keep a table matching encountered domain names with their corresponding IP address(es). We first populate the table with entries from the LAN gateway's DNS cache, and then update it whenever a packet bearing domain data is observed, i.e. a DNS query/response or a TLS `Client Hello` message with the SNI extension.[1]

2) Aggregating packets to Flow IDs: Packets that share all attributes in the set of properties below are grouped to a bidirectional flow, and the attributes form the *Flow ID* of that flow.

- The hostname (domain name or IP address) of the *source* and *destination*. The Flow ID's *source* is the source of the first packet in the bidirectional flow.
- The *protocol* (e.g. TCP or UDP).
- The source and destination *ports*. We only consider a port if it belongs to a well known service (e.g. port 80) or if it appears, for the given combination of the other attribute values, in all m^+ packet traces. This means, for example, that the packets of all TCP connections from a client A to the port 80 of server B are aggregated to a single flow and the random client ports used by A are ignored and not further considered. The same is also done for vendor-specific ports, thanks to the above rule.
- *Application*-specific data for known application layer protocols, amongst others: query name and query type for DNS; method and URI for HTTP; message type, code and URI for CoAP. Consequently, for example, a DNS response is grouped with its query (based on the query type and name).

3) Building the event signature: The aggregation of packets to Flow IDs produces a set f_i of Flow IDs, with $1 \leq i \leq m^+$, for each of the m^+ packet traces. We define the *event signature* s for the investigated interaction as the set of Flow IDs that appear in all packet traces:

$$s := \bigcap_{i=1}^{m^+} f_i$$

By only keeping the intersection, we filter out Flow IDs belonging to network communications that are not deterministically associated with the event, such as periodic or sporadic messages, as discussed in Section II and subsection III-B.

D. Step 3: Blocking flows with the event signature tree

The event signature tree is a *connected*, *acyclic*, *rooted tree* where the nodes of the tree are Flow IDs, except for the root node. The tree is used to keep track of the Flow IDs already seen and blocked.

1) Tree creation: At the beginning, when the profiling starts for a specific interaction of the device, the tree only consists of the root node. The firewall running on the switch or AP to which the Smart Home device and the companion app are connected does not block any traffic.

After obtaining the event signature s from step 2, we add each Flow ID in the signature as a child node to the root node. For the next iteration of steps 1 through 3, we select a node, i.e. a Flow ID, that has not been visited yet from the tree and

[1]We also tried reverse DNS lookups, but since most of the servers contacted during our experiments were hosted at big cloud providers such as Amazon, no meaningful data was obtained, and we abandoned this approach.

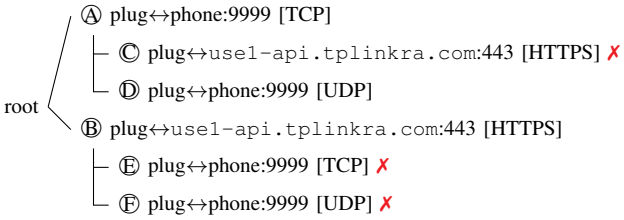

Fig. 3: *Node pruning*. The nodes Ⓒ, Ⓔ, and Ⓕ are not further explored (✗) in the BFS traversal because nodes with the same Flow ID have been already seen (nodes Ⓑ, Ⓐ, and Ⓓ, respectively).

instruct the firewall to block all packets matching that node and all its ancestors in the tree. The node will be marked as visited and the newly found Flow IDs will be added as children to it, and the procedure is repeated.

The algorithm ends if no unvisited nodes are left. Theoretically, the algorithm could continue to run indefinitely if new Flow IDs are discovered in each iteration. However, in our experiments, the algorithm always terminated after a finite number of iterations because the more Flow IDs are blocked, the more constrained the communication of the device becomes and the fewer successful event traces are captured, until no more new Flow IDs are added to the tree.

2) Tree traversal and pruning: To select the next node, we use a Breadth-First Search (BFS) [26] traversal, i.e., we process all nodes at a given depth before proceeding to the next level. Our objective is to trigger all possible network flows corresponding to a Smart Home device event, i.e., express all possible nodes in the event signature tree. To manage resource usage and execution time, two strategies are used to limit the size of the tree.

Firstly, we already remove event signatures for which less than half of the trace captures were successful, i.e., $m^+ < m/2$, in step 2. In this way, we filter out event signatures for which there is not enough data to reliably determine the Flow IDs that are with a high probability linked to the event. In practice, we observe a polarization in terms of successful event execution: either all event executions succeed ($m^+ = m$) or none ($m^+ = 0$). The latter occurs when the firewall rules prevent the event's success. Setting the threshold at $m/2$ is therefore a conservative choice.

Secondly, we prune branches of the tree which will likely not provide new information, i.e. no new Flow IDs, by applying an *ad hoc* pruning heuristic. To determine which heuristic we can apply without losing any information, we ran preliminary experiments with one of our testbed's devices, the TP-Link HS110 smart plug [27]. We observed that, for any two nodes with the same Flow ID, their children were always identical. We conclude it is likely that no new information will be found by processing a node identical to one which has already been processed, regardless of its position in the tree. Therefore, we adopt the *node pruning* heuristic, illustrated in Fig. 3: a node is pruned if an equivalent node, i.e., a node with the same Flow ID, has already been processed. Additional insight concerning this preliminary experiment is given in appendix C.

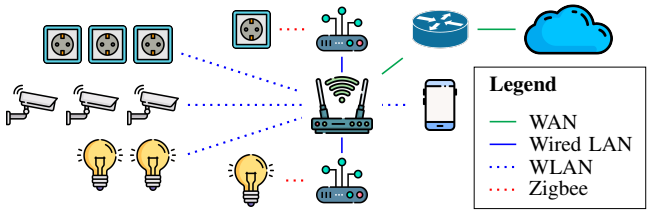

Fig. 4: Experimental Smart Home network.

3) How to block packets matching a Flow ID: As explained, a Flow ID comprises traffic features from multiple networking layers, such as hostnames, port numbers, and application layer information. The presence of application layer information in the Flow ID requires a firewall that is able to match and reject traffic based on information found at that layer. A simple firewall, such as the default Linux kernel firewall NFTables [28] (successor of the well-known IPTables) is not sufficient. Instead, we leverage the open-source Smart Home firewall we developed as previous work [9]. It is based on NFTables, while enhancing its capabilities to match additional application-layer protocols. However, this firewall can only work with *allow* rules, while our profiling algorithm is based on blocking rules. Therefore, we modified the firewall, turning it into a *deny-list* firewall. In our prototype implementation of the profiling algorithm, we translate the list of Flow IDs into firewall blocking rules and let the firewall enforce them for the next batch of experiments. The code of the modified firewall is available at https://github.com/smart-home-network-security/firewall-blocklist.

IV. Experimental Results

In this section, we present our experimental setup, and the results obtained from applying our framework to it.

A. Experiment setup

We apply our methodology to a controlled testbed network comprised of real-world devices, mimicking a typical Smart Home network, depicted in Fig. 4. We examine an array of commercial, off-the-shelf Smart Home devices, including four power plugs, three cameras, and three light bulbs. The profiled interactions, per device category, are the following:

- For all devices: booting the device;
- Power plugs: toggling them on/off;
- Cameras: streaming their video feed;
- Light bulbs: toggling them on/off, changing their brightness, changing their color.

In a real Smart Home network, a user could choose among various apps to control their devices, including the device's official companion app (from its manufacturer or vendor), a third-party app, or a Smart Home automation platform which provides unified control over devices from different

Device	App	Interaction	Event ID
Power plugs			
TP-Link HS110 [27]	/	boot	1
	Kasa Smart	toggle	11
	TP-Link Tapo	toggle	12
	SmartThings	toggle	13
SmartThings Outlet † [33]	/	boot	2
	SmartThings	toggle	14
Tapo P110 [34]	/	boot	3
	TP-Link Tapo	toggle	15
	SmartThings	toggle	16
Woox R5024 (Tuya) [35]	/	boot	4
	Tuya Smart	toggle	17
Cameras			
Xiaomi MJSXJ02CM [36]	/	boot	5
	Mi Home	stream	18
Tapo C200 [37]	/	boot	6
	TP-Link Tapo	stream	19
	SmartThings	stream	20
D-Link DCS-8000LH [38]	/	boot	7
	mydlink	stream	21
Light bulbs			
Philips Hue White and Color lamp A60 † [39]	/	boot	8
	Philips Hue	toggle	22
	Hue Essentials	toggle	23
	Hue Essentials	brightness	24
	Hue Essentials	color	25
	SmartThings	toggle	26
	SmartThings	brightness	27
	SmartThings	color	28
Alecto Smart-Bulb10 (Tuya) [40]	/	boot	9
	Tuya Smart	toggle	29
	Tuya Smart	color	30
Tapo L530E [41]	/	boot	10
	TP-Link Tapo	toggle	31
	TP-Link Tapo	brightness	32
	TP-Link Tapo	color	33
	SmartThings	toggle	34
	SmartThings	brightness	35
	SmartThings	color	36

TABLE I: Devices in our testbed, with corresponding apps and interactions. † indicates Zigbee devices, for which the hub traffic was considered.

manufacturers. Such different home automation methods can trigger diverging, but valid, communication patterns. To cover such cases, for all interactions other than booting, we derive up to three usage scenarios:

- For all devices, we issue their interactions using their official companion app;
- For devices that support it, we also control them using the app of the popular home automation platform *SmartThings* [29];
- For the TP-Link plug and the Hue light bulb, we also experiment with other apps; respectively, *TP-Link Tapo* [30] (another app from the same manufacturer, targeted toward *Tapo*-branded devices) and *Hue Essentials* [31] (third-party app for Hue devices).

The smartphone running the device-specific apps is a Crosscall CORE-X4 [32] with the Android 10 OS.

The complete device references are listed in Table I, along with the corresponding smartphone apps and interactions instrumented. The numerical identifier used as x-axis for some subsequent plots pertains to the *Event ID* listed in the table,

Fig. 5: Event signature tree for the TP-Link HS110's *toggle* event. Bidirectional flows are represented by a double-headed arrow, whilst a single-headed arrow is used for unidirectional flows. When indicated, the port is associated with its host.

which is a unique number assigned by us to each of the tested combinations of device, app, and interaction. As explained, we only consider IP traffic, whether over Ethernet or Wi-Fi. Therefore, for the two devices communicating using Zigbee, we profile the traffic issued by their hub, i.e. the Hue bridge for the Hue lamp, and the SmartThings hub for the SmartThings Outlet, respectively. For the *boot* interaction experiments over those devices, we synchronously power-cycle both the device itself and its hub.

Our practical experimental parameters, inspired by related works *PingPong* [11] and *IoTAthena* [15], are the following:

- Number of event iterations: $m = 20$;
- Traffic capture timeout: $d = 20$ seconds (90 seconds for interaction *boot*);
- Duration between two event iterations: randomly chosen between 40 (120 for interaction *boot*) and 150 seconds.

B. Example signature tree

As an example of our framework's results, we present the event signature tree generated for one device of our corpus, namely the TP-Link HS110 [27]. This device being a smart power plug, it provides one interaction, i.e. toggling it on and off. For this experiment, we controlled the device using its official Android companion app, i.e. Kasa Smart [42]. The resulting signature tree is displayed in Fig. 5. To avoid unnecessarily cluttering the figure, we only show the first occurrence of each unique flow. We empirically observe that each one of the listed *first-level* flows also occurs as a child of every other node in the tree, i.e. the communication pattern is relatively stable. In certain cases, a *hidden* flow will appear at deeper levels, which embodies a backup communication strategy when the default one fails.

By default, toggling the plug triggers seven different flows:

Ⓐ between the phone and the plug's TCP port 9999;
Ⓑ between the phone and the plug's UDP port 9999;
Ⓒ from the phone to the mDNS multicast address (224.0.0.251, UDP port 5353);

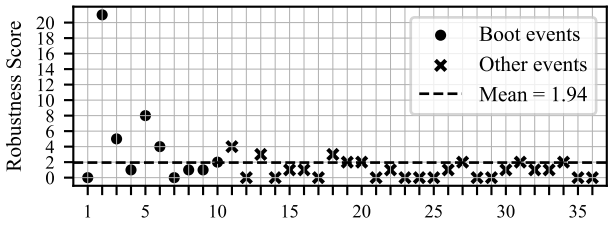

Fig. 6: *Robustness score*. The x-axis shows the *Event ID*.

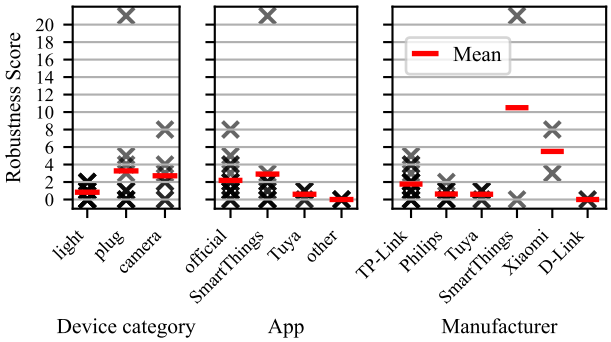

Fig. 7: *Robustness score* for all device interactions, grouped per device type, app, and manufacturer. Each marker represents the robustness score of one event.

Ⓓ from the phone to the broadcast address' (255.255.255.255) UDP port 9999;
Ⓔ HTTPS between the phone and the server `xx-device-telemetry-gw.iot.i.tplinknbu.com`;
Ⓕ DNS query/response from the plug to the LAN gateway, for the domain name `use1-api.tplinkra.com`;
Ⓖ HTTPS between the plug and the server `use1-api.tplinkra.com`.

Flows Ⓒ and Ⓓ are unidirectional, while the remaining ones are bidirectional.

When one of those flows is blocked, the device might issue different network traffic to perform its event. Indeed, when the TCP communication Ⓐ on the plug's port 9999 is blocked, we see two new HTTPS flows appearing: between the phone and the server `n-wap.tplinkcloud.com`, and between the plug and the IPv4 address `79.125.56.92`. The former also occurs when other first-level flows are blocked, including the similar pattern Ⓑ using UDP instead of TCP. Both flows are part of a backup strategy, going through the internet, if the device is not able to communicate locally with the controlling phone. Such *hidden* backup flows also appear as children of the *first-level* flow Ⓓ.

C. Assessing device event robustness

The experiments in our testbed of devices result in the extraction of 254 unique Flow IDs, of which 70 are *hidden* (i.e. 27.56%). Their distribution over the 36 tested event configurations is shown in Fig. 11 in the appendix. *Hidden* Flow IDs embody the alternative communication strategies that a device might use if the default one fails, effectively enhancing the device's robustness to network instability (remember that we only keep Flow IDs associated with successful event executions). We define the robustness score as the count of hidden Flow IDs. Recall that a *hidden* Flow ID is one which only appears in the tree deeper than the first level, i.e. resulting from blocking specific flows. We compute this metric for each tested event; Fig. 6 shows the results.

Out of the 36 instrumented events, 23 (63.9%) have a robustness score of at least one (mean 1.94). This value is encouraging, as it means that most types of interactions dispose of at least one backup strategy in the case of a failure.

To gain further insight into the distribution of robustness scores, Fig. 7 shows the same *robustness score*, grouped along three axes, namely:

- Device category;
- Controlling app;
- Manufacturer.

We identify an event with a robustness score significantly higher than all others: the *boot* event for the SmartThings Outlet device. Being a Zigbee-communicating device, its *boot* event consists in switching on the plug itself as well as its related hub. As hubs are commonly used to provide centralized control over a Smart Home network, they must provide a way to detect the devices present in the network. It is expected that, if the default way fails, the hub will follow backup communication paths to provide this functionality.

We observe that the different device categories in our testbed do not provide the same robustness. The most robust devices seem to be the power plugs. A rationale might be that, as the plugs' operation (switching on or off) is the simplest, it takes less effort to the manufacturers to provide backup strategies. However, given the size of the testbed compared to the vast number of devices on the market, this observation should be taken as indicative only. This also applies to the following discussion of the companion apps and manufacturers.

Among the companion apps, Tuya seems to be less robust than the others. While most apps provide device control either via the local network or through the cloud, Tuya devices are notoriously reluctant to LAN-based control, preferring cloud-based communication. This decision reduces their robustness by design, as, when the cloud endpoints cannot be reached, the device cannot function properly. SmartThings boasts a good score, both as a controlling app and a manufacturer, helped by a very high robustness score for the SmartThings Outlet's *boot* interaction. We can also see that the "other" app category provides no backup strategy whatsoever, suggesting that a user should prefer using trusted and official apps, even if the devices' API is theoretically open.

We illustrate the effect of the choice of the app with the Philips Hue lamp device and its *toggle* interaction, which we have triggered by the official app (Philips Hue), the SmartThings app, and a third-party app (Hue Essentials). The

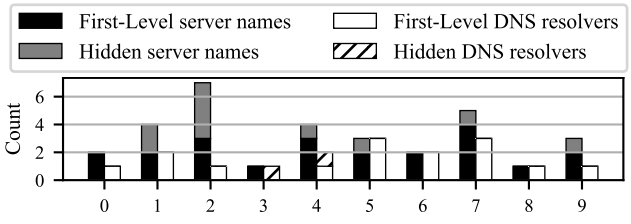

Fig. 8: Domain names / DNS resolvers contacted for each *boot* event. The x-axis shows the *Event ID*.

official app and SmartThings show similar results: five and six *first-level* Flow IDs respectively, and one *hidden* flow for each. However, when using the Hue Essentials app, only three *first-level* flows are extracted, and no *hidden* flow.

The Xiaomi camera also stands out with a high mean score. As cameras are usually tied to more critical functions, such as monitoring a home against potential intrusions, Xiaomi followed a respectable rationale here. TP-Link seems quite robust, while Philips, even as a trusted device manufacturer, disappoints in terms of robustness.

D. A closer look at DNS data

The Domain Name System (DNS), which allows communicating with hosts by providing a domain name instead of their IP address, is a useful tool to provide application robustness. Here, we steer our analysis towards interactions with the DNS protocol, guided by two questions:

- If the servers the device tries contacting are unresponsive, will it contact servers with other domain names?
- If the device's default DNS resolver is unreachable, will the device try other ones?

We focus on event signatures related to *boot* interactions, as those proved to provide the most information concerning DNS usage. Fig. 8 shows the data related to both questions: the count of unique domain names contacted, and that of DNS resolvers queried. In both cases, the *first-level* [2] and the *hidden* count are shown.

1) Server's domain names: We observe that, out of the 10 *boot* events, six consider contacting at least one alternative domain name if the default one fails.

2) DNS resolvers: Out of the 17 DNS resolvers contacted, only two are *hidden* resolvers. One of them is simply the LAN gateway (`192.168.1.1`), meaning the only real new DNS resolver discovered at a level deeper than one is the Chinese public DNS resolver `114.114.114.114`, contacted by the Xiaomi camera when the LAN gateway is not responsive. This result shows that, regarding domain name resolving, Smart Home devices still lack robustness. Indeed, every device should try contacting a backup resolver if the primary one fails, as this has been made easy thanks to publicly available resolvers such as Cloudflare's `1.1.1.1` or Google's `8.8.8.8`.

[2]In this context, the qualifier *first-level* indicates the occurrences at the signature tree's first depth level; it has no relation with the similarly named, but unrelated, domain names' top-level domain.

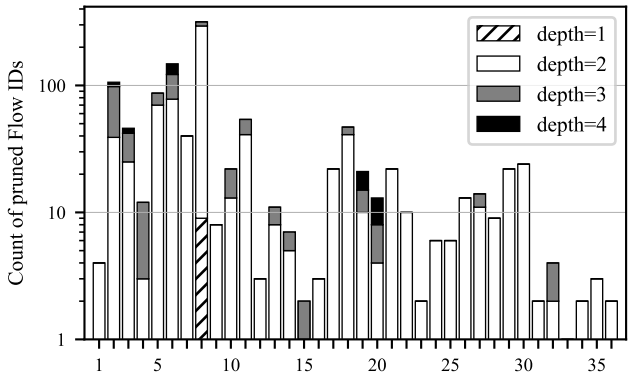

Fig. 9: Pruned Flow IDs. The x-axis shows the *Event ID*.

E. Analysis of the node pruning heuristic

We analyze the effect of our tree pruning heuristic described in Section III-D2. Fig. 9 illustrates, for every studied event, the count of Flow IDs which have been pruned while generating the event signature tree, split per node depth. As a reminder, this heuristic states that we prune a node, i.e. we do not process the children of a node, if an equivalent node is already present in the tree; the depicted count is therefore equal to the number of duplicate Flow IDs in the tree.

We observe that the count is generally important. In terms of efficiency, this means that our heuristics manages to significantly reduce the time and resources taken by the event signature tree generation.

We also notice that most of the duplicate Flow IDs occur at depth 2. Those represent all the flows that occur after blocking a first-level flow. Their large number shows that most devices exhibit a high redundancy in communication patterns. Indeed, if the device must change its network behavior to circumvent a traffic restriction, it will usually only change the affected protocol, e.g. by contacting another domain name or switching from TCP to UDP; it will not completely modify the communication pattern.

F. Comparison with related work

We compare the profiles generated by our framework with those generated by methods proposed in related work, namely *PingPong* [11], *BehavIoT* [14], and *MUDgee* [43], for the devices present in both their testbeds and ours. *PingPong* and *BehavIoT* profile specific interactions of a device, whereas *MUDgee* profiles the device as a whole. Table II summarizes the devices and interactions covered by each work, as well as the smartphone apps which produced the subset of our data used for our side in the comparison.

Fig. 10 compares the number of unique Flow IDs discovered by each work and by us. Data from *PingPong* and *MUDgee* were taken from their reported results, whereas *BehavIoT*'s results were reproduced on our side. The three related works, by design, do not explore the event signature tree deeper than the first level. Regarding the specific interactions, our approach

Devices	Apps	PingPong [43]	BehavIoT [11]	MUDgee [14]
TL plug	Kasa Smart	toggle	toggle	device
Hue lamp	Philips Hue	toggle	✗	device
ST plug	SmartThings	toggle	boot	device
DL cam	mydlink	✗	stream	✗

TABLE II: Related works' device and interaction coverage. Device legend: TL plug = TP-Link HS110; Hue lamp = Philips Hue lamp; ST plug = SmartThings Outlet; DL cam = D-Link camera.

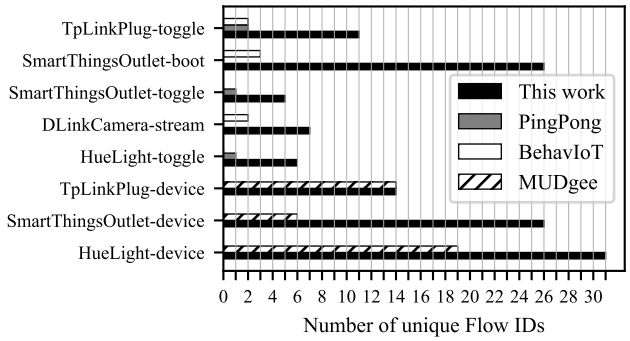

Fig. 10: Comparison of discovered unique Flow IDs between three related works and ours. Note that MUDgee profiles devices as a whole and not for individual interactions.

is, as expected, able to extract more Flow IDs than *PingPong* and *BehavIoT*, thanks to a deeper tree inspection, and also a more thorough communication pattern extraction, even at the first level. To compare our results with the device-level profiles created by *MUDgee*, we count the unique Flow IDs over all interactions of a device. For two of the instrumented devices, namely the SmartThings Outlet and the Hue light, our approach is again able to extract a significantly higher number of Flow IDs than *MUDgee*. However, the number of Flow IDs is identical for the TP-Link HS110. A manual investigation shows that the extracted Flow IDs are different: 10 out of 14 *MUDgee* flows are NTP traffic towards various servers, whereas our solution did not extract any NTP patterns. This traffic might be part of periodic patterns, which are not covered by our approach; as explained in Section III-B, our methodology excludes traffic that does not appear in every iteration of a tested interaction from the profile building process. On the other hand, we extract 10 non-NTP traffic patterns which were not detected by *MUDgee*.

Additionally, we compare the number of domains names discovered by our approach with those reported by Mandalari *et al.* [24]. The latter focuses on extracting the hosts from Smart Home communication patterns. Table III shows, in the columns "Total ours" and "Total theirs", the total number of unique domain names discovered by our approach and by theirs for the devices and interactions covered in both works. We observe that our multi-level approach is able to discover more domain names. However, it should be noted that changes in the firmware of the devices since the publication of the cited article [24] could influence the results.

A direct comparison of the domain names beyond the mere numbers is unfortunately not possible. This is because Mandalari *et al.* were primarily interested in finding "non-essential" hosts, i.e. hosts that can be blocked, for example for privacy reasons, without negatively impacting the essential functionality of an IoT device. Consequently, while the total numbers are known, they only published the domain names of non-essential hosts, whereas we do not differ between essential and non-essential ones. Nevertheless, we have also included the actual domain names in Table III. The differences between the two works can be explained by the fact that our approach actively looks for hidden communication patterns, the fact that their work is focused on identifying non-essential hosts, and the reasons already mentioned above. For example, the domain name `ecdinterface.philips.com` does not exist anymore (in 2025), and the domain name `diagnostics.meethue.com` could be part of a periodic or sporadic pattern filtered out by our approach. This can also explain the empty intersections between the two works for two of the tested devices.

V. Related Work

Research on IoT device profiling has been fruitful in the last years. Such works lie at differing layers of abstraction, which can be classified along two orthogonal axes:

- *Object abstraction*: which semantic unit is profiled, from individual events, to single devices, to the IoT network as a whole.
- *Behavior abstraction*: whether the work intends to profile the low-level network traffic, which can be further dissected into individual packets or aggregate flows, or the higher-level abstract device events retrieved from home automation systems.

In the following, we will summarize related works, and position them along the aforementioned abstraction axes; Table IV shows the resulting classification.

Girish *et al.* [4] conducted the first comprehensive analysis of LAN Smart Home traffic. They exhaustively characterized the protocols used, highlighting the security and privacy vulnerabilities, of intra-network device communication.

Among the numerous proposals for IoT device behavior models, the most widespread is probably the IETF's MUD standard [6], as it has been standardized and backed up by multiple big players, including Cisco [46]. Subsequently, research leveraging this standard was fueled, including Hamza *et al.*'s solution to generate MUD profiles from network traces of IoT devices' traffic [10]. The generated profiles, however, suffer from the shortcomings inherent to the MUD standard, namely, among others, the lack of support for protocols other than IP (v4 & v6) on layer 3, and TCP, UDP and ICMP on layer 4. In previous work [9] proposed a syntax to express the intended network behavior of devices, inspired by MUD, while overcoming some of its shortcomings, and including support for traffic related to cross-device interactions.

Device & interact.	Total ours	Total theirs	Both	Ours only	Theirs only
TP-Link HS110 boot & toggle	5	4	use1-api.tplinkra.com euw1-api.tplinkra.com	euw1-device-telemetry-gw.iot.i.tplinknbu n-wap.tplinkcloud.com n-devs.tplinkcloud.com	n-deventry.tplinkcloud.com
SmartThings Outlet boot	4	3	/	hub.smartthings.com usher.connect.smartthings.com connectivity.smartthings.com dc2-eu01-euwest1.connect.smartthings.com	fw-update2.smartthings.com
Philips Hue lamp boot	5	4	/	mqtt-eu-01.iot.meethue.com time.meethue.com ntp2.aliyun.com ntp4.aliyun.com time4.google.com	diagnostics.meethue.com ecdinterface.philips.com

TABLE III: Domain names extracted by Mandalari *et al.* [24] (referred to as "theirs") and our work (referred to as "ours"). Note that the domain names reported by Mandalari *et al.* only concern non-essential hosts.

	Event	Device	Network
Packets	*PingPong* [11], *D-interact* [44]	*IoTa* [45]	Previous work [9]
Flows	Mandalari [24], **This work**	MUD [6], Hamza [10]	Previous work [9], Girish [4]
Events	/	*BehavIoT* [14]	

TABLE IV: Abstraction level of device behavior models in our work and in the literature

Regarding research efforts to extract event signatures from network traffic, three works follow an interact & measure workflow similar to ours: *PingPong* [11], *D-interact* [44], and *IoTa* [45]. Nevertheless, all three lie at a finer granularity level than our work, as their signatures are composed of individual packets, and leverage packet-level metadata, such as the packet size. We argue that such features can be subject to network churn and therefore vary from one event execution to the other, making them imprecise to accurately characterize network behavior.

Mandalari *et al.* [24] centered their analysis on the hosts contacted by Smart Home devices, characterized mainly by their domain names, but falling back to their plain IP address if the domain name was unavailable. Their goal was to identify non-required hosts, i.e. hosts which could be blocked without impairing the device's operation, e.g. telemetry or advertisement services. While they designed a workflow similar to ours to identify the successful events, they did not further explore the event signature tree.

Hu *et al.* proposed *BehavIoT* [14], a system which models the behavior of a whole Smart Home network, with the intent of using it to detect when the network produces unintended behavior, potentially due to unwanted communication, or even a security breach. Their model is twofold: on the one hand, they model individual devices events; on the other hand, cross-device interactions. Their complete model is generated based on the network traffic produced by the devices, and takes the form of a probabilistic Finite State Machine (FSM), representing all the possible event transitions in the home network.

All the above-mentioned works have in common that, unlike our work, they do not actively attempt to discover hidden communication patterns.

Finally, the concept of chaos engineering [47] is related to our approach. It consists in injecting faults (e.g. network loss or latency) or heavy load (e.g. on CPU, memory, etc.). While our approach shares some concepts with chaos engineering, its novelty resides in the fact that we are the first to apply it to this extent in the context of smart homes. Moreover, today, no chaos engineering tool provides control over network traffic at a protocol level as precise as us, by blocking traffic based on specific fields. The most precise is Steadybit [48], which allows blocking TCP altogether.

VI. Conclusion

In this article, we presented a novel methodology and framework to extract comprehensive, multi-level traffic signatures from Smart Home device events, whereas existing works are limited to first-level signatures. The signatures take the form of trees, with the nodes being the constitutive traffic flows. By the usage of a dynamic blocking scheme, our approach allows uncovering *hidden* traffic patterns that only occur when the default communication mechanisms of a device fail.

We show that more than half of the tested devices exhibit such hidden traffic patterns. In average, two hidden flows are discovered for each device, across all studied interactions with the device. Our work highlights the robustness capabilities of Smart Home devices, by proposing a new metric dubbed the *robustness score*. We can envision that, in the future, such a score might be used as a marketing argument; indeed, customers would prefer a device which is able to function even in inhospitable network conditions.

As future work, we would like to apply our methodology to scenarios where multiple Smart Home devices are involved in a single interaction. This is conceptually supported by our framework, but will require an extension of the mechanisms we use for event triggering. In this case, approaches to further automate the latter should also be investigated. Besides, our robustness analysis only focused on binary traffic verdict, i.e. either the traffic passes or is blocked. To further extend our robustness analysis, we could consider other traffic disturbances, such as limiting the traffic rate and inserting traffic bursts. A study covering such bursts would be transversal to ours, focusing more on robustness with respect to, e.g., DoS attacks.

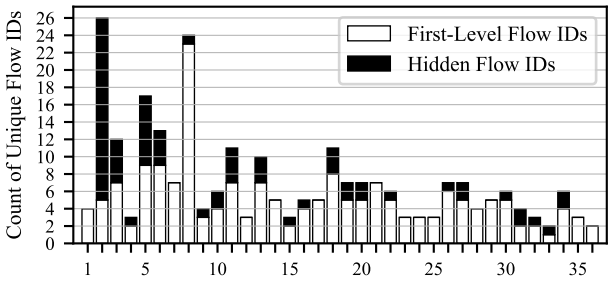

Fig. 11: Unique Flow IDs. The x-axis shows the *Event ID*.

ACKNOWLEDGMENTS

F. De Keersmaeker is funded by the F.R.S.-FNRS Research Fellow Renewal [ASP-REN] grant with number 40017864. R. Sadre is supported by the Walloon Belgian region project CyberExcellence. C. Pelsser is supported partially by grant #1318167 from the Cisco University Research Program Fund, an advised fund of Silicon Valley Foundation, and the Win4Collective CARAPACE project. All icons are from flaticon.com.

APPENDIX

A. Statement on ethical issues

No ethical issues were raised by this work. Our device profiling activities are in line with the EU legislation on reverse engineering [49], [50]. All experiments were performed in a controlled environment, and did not involve any user personal data or other privacy concerns.

B. List of protocol fields supported by the firewall

Table V displays all the protocol fields which can be leveraged by the firewall to block packets.

C. Event signature tree size without pruning

In this appendix, we give additional insight advocating for our event signature tree pruning heuristic. We generated a sample tree, by performing preliminary experiments with the *toggle* interaction of the TP-Link HS110 [27]. The size of the complete tree was huge: it comprised 75 Flow IDs in total. However, among these, only five were unique. As explained in the main text (Section III-D2), each occurrence of the same node had the same set of children Flow IDs. We deem our *node pruning* heuristic will not lose information, and is efficient in reducing the tree to a manageable size.

D. Additional results

Fig. 11 shows the count of unique Flow IDs discovered by our framework for each investigated event, split into *first-level* and *hidden* Flow IDs. The x-axis pertains to the event ID, as provided in Table I.

REFERENCES

[1] P. Sethi and S. R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications," *Journal of Electrical and Computer Engineering*, vol. 2017, no. 1, p. 9324035, 2017.

[2] (2024, 11) Smart Home Market Analysis - 2032. Fortune Business Insights. Accessed: December 2, 2024. [Online]. Available: https://www.fortunebusinessinsights.com/industry-reports/smart-home-market-101900

[3] 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.

[4] A. Girish, T. Hu, V. Prakash, D. J. Dubois, S. Matic, D. Y. Huang, S. Egelman, J. Reardon, J. Tapiador, D. Choffnes, and N. Vallina-Rodriguez, "In the room where it happens: Characterizing local communication and threats in smart homes," in *Proceedings of the 2023 ACM on Internet Measurement Conference*, ser. IMC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 437–456.

[5] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, Aug. 2019, conference Name: IEEE Transactions on Mobile Computing.

[6] E. Lear, R. Droms, and D. Romascanu, "Manufacturer Usage Description Specification," RFC 8520, Mar. 2019.

[7] S. N. Matheu, J. L. Hernández-Ramos, S. Pérez, and A. F. Skarmeta, "Extending MUD Profiles Through an Automated IoT Security Testing Methodology," *IEEE Access*, vol. 7, pp. 149 444–149 463, 2019.

[8] S. Singh, A. Atrey, M. L. Sichitiu, and Y. Viniotis, "Clearer than Mud: Extending Manufacturer Usage Description (MUD) for Securing IoT Systems," in *Internet of Things – ICIOT 2019*, ser. Lecture Notes in Computer Science, V. Issarny, B. Palanisamy, and L.-J. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 43–57.

[9] F. De Keersmaeker, R. Sadre, and C. Pelsser, "Supervising Smart Home Device Interactions: A Profile-Based Firewall Approach," in *2024 IFIP Networking Conference (IFIP Networking)*, June 2024, pp. 413–422.

[10] A. Hamza, D. Ranathunga, H. H. Gharakheili, M. Roughan, and V. Sivaraman, "Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, ser. IoT S&P '18. New York, NY, USA: Association for Computing Machinery, Aug. 2018, pp. 8–14.

[11] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-Level Signatures for Smart Home Devices," in *Proceedings 2020 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2020.

[12] 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. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 128–138.

[13] 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, Oct. 2020, pp. 87–100.

[14] T. Hu, D. J. Dubois, and D. Choffnes, "BehavIoT: Measuring Smart Home IoT Behavior Using Network-Inferred Behavior Models," in *Proceedings of the 2023 ACM on Internet Measurement Conference*. Montreal QC Canada: ACM, Oct. 2023, pp. 421–436.

[15] Y. Wan, K. Xu, F. Wang, and G. Xue, "IoTAthena: Unveiling IoT Device Activities From Network Traffic," *IEEE Transactions on Wireless Communications*, vol. 21, no. 1, pp. 651–664, Jan. 2022, conference Name: IEEE Transactions on Wireless Communications.

[16] (2024) Zigbee | Complete IoT Solution. Connectivity Standards Alliance. Accessed: December 6, 2024. [Online]. Available: https://csa-iot.org/all-solutions/zigbee

[17] I. Unwala, Z. Taqvi, and J. Lu, "Thread: An IoT Protocol," in *2018 IEEE Green Technologies Conference (GreenTech)*, 2018, pp. 161–167.

[18] P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards," *Computer Communications*, vol. 30, no. 7, pp. 1655–1695, 2007, wired/Wireless Internet Communications.

Layer	Protocol	Field	Description
Network	IPv4	src	Source host (IP address or domain name)
	IPv6	dst	Destination host (IP address or domain name)
Transport	TCP	src-port	Source port (linked to source host)
	UDP	dst-port	Destination port (linked to destination host)
Application	(m)DNS	qtype	Query type (e.g. A, AAAA, CNAME, etc.)
		qname	Queried domain name
	HTTP	is_response	Whether the packet is an HTTP response
		method	HTTP method (e.g. GET, PUT, POST, etc.)
		uri	Requested URI
	CoAP	is_response	Whether the packet is a CoAP response
		code	CoAP request code (e.g. GET, PUT, POST, etc.)
		uri_path	Requested URI

TABLE V: List of protocol fields, per layer, which can be leveraged by the firewall for its verdict

[19] C. W. Badenhop, S. R. Graham, B. W. Ramsey, B. E. Mullins, and L. O. Mailloux, "The Z-Wave routing protocol and its security implications," *Computers & Security*, vol. 68, pp. 112–129, 2017.

[20] M. A. M. Almuhaya, W. A. Jabbar, N. Sulaiman, and S. Abdulmalek, "A Survey on LoRaWAN Technology: Recent Trends, Opportunities, Simulation Tools and Future Directions," *Electronics*, vol. 11, no. 1, 2022.

[21] M. H. Mazhar and Z. Shafiq, "Characterizing Smart Home IoT Traffic in the Wild," in *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2020, pp. 203–215.

[22] (2024, 9) Android Debug Bridge (adb). Google. Accessed: December 4, 2024. [Online]. Available: https://developer.android.com/tools/adb

[23] (2025, 12 May) python-kasa 0.10.2. Python Software Foundation. Accessed: May 12, 2025. [Online]. Available: https://pypi.org/project/python-kasa

[24] A. M. Mandalari, D. J. Dubois, R. Kolcun, M. T. Paracha, H. Haddadi, and D. Choffnes, "Blocking Without Breaking: Identification and Mitigation of Non-Essential IoT Traffic," in *Proceedings on Privacy Enhancing Technologies*, 2021.

[25] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[26] R. Sedgewick and K. Wayne, *Algorithms*. Addison-Wesley Professional, 2011.

[27] (2024) HS110 | Kasa Smart Wi-Fi Plug with Energy Monitoring. TP-Link. Accessed: December 5, 2024. [Online]. Available: https://www.tp-link.com/en/home-networking/smart-plug/hs110

[28] (2023, 23 Aug.) The netfilter.org "nftables" project. Netfilter. Accessed: August 24, 2023. [Online]. Available: https://www.netfilter.org/projects/nftables/index.html

[29] (2024) Do the SmartThings! Control your home with ease. Samsung. Accessed: December 2, 2024. [Online]. Available: https://www.samsung.com/us/smartthings

[30] (2025) TP-Link Tapo. TP-Link Systems Inc. Accessed: March 21, 2025. [Online]. Available: https://play.google.com/store/apps/details?id=com.tplink.iot

[31] (2025) Hue Essentials. SmartFusionLabs. Accessed: March 21, 2025. [Online]. Available: https://play.google.com/store/apps/details?id=com.superthomaslab.hueessentials

[32] (2024) CORE-X4: professional, efficient and practical smartphone. TP-Link. Accessed: December 5, 2024. [Online]. Available: https://www.crosscall.com/en_GB/core-x4-COX4.MASTER.html

[33] (2024) Smart Plug (2019). Samsung. Accessed: December 5, 2024. [Online]. Available: https://www.samsung.com/uk/smartthings/outlet/smartthings-smart-plug-gp-wou019bbdwg

[34] (2024) Tapo P110 | Mini Smart Wi-Fi Socket, Energy Monitoring. TP-Link. Accessed: December 5, 2024. [Online]. Available: https://www.tp-link.com/en/home-networking/smart-plug/tapo-p110/v1

[35] (2024) Woox R5024. Woox. Accessed: December 5, 2024. [Online]. Available: https://wooxhome.com/products-c10/power-c2/woox-r5024-smart-plug-eu-p8

[36] (2024) Mi Home Security Camera 360°. Xiaomi. Accessed: December 5, 2024. [Online]. Available: https://www.mi.com/global/camera-360

[37] (2024) Tapo C200 | Pan/Tilt Home Security Wi-Fi Camera. TP-Link. Accessed: December 5, 2024. [Online]. Available: https://www.tp-link.com/en/home-networking/cloud-camera/tapo-c200

[38] (2024) DCS-8000LH Mini HD WiFi Camera. D-Link. Accessed: December 5, 2024. [Online]. Available: https://www.dlink.com/uk/en/products/dcs-8000lh-mini-hd-wifi-camera

[39] (2024) Hue White and Color Ambiance A60 - E27 smart bulb - 800. Philips. Accessed: December 5, 2024. [Online]. Available: https://www.philips-hue.com/en-my/p/hue-white-and-color-ambiance-a60---e27-smart-bulb---800/8718696725580

[40] (2024) Alecto SMART-BULB10 DUO - Smart LED colour lamp with Wi-Fi, 2 pack. Alecto Home. Accessed: December 5, 2024. [Online]. Available: https://alectohome.com/products/alecto-smart-bulb10-duo-smart-wifi-led-lamp-2-pack-wit

[41] (2024) Tapo L530E | Smart Wi-Fi Light Bulb, Multicolor. TP-Link. Accessed: December 5, 2024. [Online]. Available: https://www.tp-link.com/en/home-networking/smart-bulb/tapo-l530e

[42] (2025) Kasa Smart. TP-Link Systems Inc. Accessed: March 21, 2025. [Online]. Available: https://play.google.com/store/apps/details?id=com.tplink.kasa_android

[43] A. Hamza, D. Ranathunga, H. H. Gharakheili, T. A. Benson, M. Roughan, and V. Sivaraman, "Verifying and Monitoring IoTs Network Behavior Using MUD Profiles," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 1–18, 2022.

[44] M. Sun, K. Li, Y. Zheng, W. Zhang, H. Li, and L. Sun, "Inferring Device Interactions for Attack Path Discovery in Smart Home IoT," in *Wireless Algorithms, Systems, and Applications*, L. Wang, M. Segal, J. Chen, and T. Qiu, Eds. Cham: Springer Nature Switzerland, 2022, pp. 64–75.

[45] C. Duan, S. Li, H. Lin, W. Chen, G. Song, C. Li, J. Yang, and Z. Wang, "IoTa: Fine-Grained Traffic Monitoring for IoT Devices via Fully Packet-Level Models," *IEEE Transactions on Dependable and Secure Computing*, no. 01, pp. 1–17, Dec. 2023, publisher: IEEE Computer Society.

[46] (2024) What is MUD? Cisco DevNet. Accessed: November 21, 2024. [Online]. Available: https://developer.cisco.com/docs/mud/what-is-mud

[47] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos Engineering," *IEEE Software*, vol. 33, no. 3, pp. 35–41, 2016.

[48] (2025, 7) Steadybit |chaos engineering tool for reliability testing. Steadybit. Accessed: July 24, 2025. [Online]. Available: https://steadybit.com

[49] European Union, "Directive 2009/24/ec of the european parliament and of the council," *Official Journal of the European Union*, vol. L 111, pp. 16–22, 5 May 2009.

[50] A. Vidstrom. (2019, 19 May) The legal boundaries of reverse engineering in the EU. Vidstrom Labs. Accessed: December 6, 2024. [Online]. Available: https://vidstromlabs.com/blog/the-legal-boundaries-of-reverse-engineering-in-the-eu