Verifying and Monitoring IoTs Network Behavior using MUD Profiles

Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili,
Theophilus A. Benson, Matthew Roughan, and Vijay Sivaraman

Abstract—IoT devices are increasingly being implicated in cyber-attacks, raising community concern about the risks they pose to critical infrastructure, corporations, and citizens. In order to reduce this risk, the IETF is pushing IoT vendors to develop formal specifications of the intended purpose of their IoT devices, in the form of a Manufacturer Usage Description (MUD), so that their network behavior in any operating environment can be locked down and verified rigorously.

This paper aims to assist IoT manufacturers in developing and verifying MUD profiles, while also helping adopters of these devices to ensure they are compatible with their organizational policies and track device network behavior using their MUD profile. Our first contribution is to develop a tool that takes the traffic trace of an arbitrary IoT device as input and automatically generates the MUD profile for it. We contribute our tool as open source, apply it to 28 consumer IoT devices, and highlight insights and challenges encountered in the process. Our second contribution is to apply a formal semantic framework that not only validates a given MUD profile for consistency, but also checks its compatibility with a given organizational policy. We apply our framework to representative organizations and selected devices, to demonstrate how MUD can reduce the effort needed for IoT acceptance testing. Finally, we show how operators can dynamically identify IoT devices using known MUD profiles and monitor their behavioral changes in their network.

Index Terms—IoT, MUD, Policy Verification, Device Discovery, Compromised Device Detection

1 Introduction

The Internet of Things is considered the next technological mega-trend, with wide reaching effects across the business spectrum [2]. By connecting billions of every day devices from smart watches to industrial equipment to the Internet, IoT integrates the physical and cyber worlds, creating a host of opportunities and challenges for businesses and consumers alike. But, increased interconnectivity also increases the risk of using these devices.

Many connected IoT devices can be found on search engines such as Shodan [3], and their vulnerabilities exploited at scale. For example, Dyn, a major DNS provider, was subjected to a DDoS attack originating from a large IoT botnet comprising thousands of compromised IP-cameras [4]. IoT devices, exposing TCP/UDP ports to arbitrary local endpoints within a home or enterprise, and to remote entities on the wider Internet, can be used by inside and outside attackers to reflect/amplify attacks and to infiltrate otherwise secure networks [5]. IoT device security is thus a top concern for the Internet ecosystem.

These security concerns have prompted standards bodies to provide guidelines for the Internet community to build secure IoT devices and services [6]–[8], and for regulatory bodies (such as the US FCC) to control their use [9]. The focus of our work is an IETF standard called Manufacturer Usage Description (MUD) [10] which provides the first formal framework for IoT behavior that can be rigorously enforced. This framework requires manufacturers of IoTs to publish a behavioral profile of their device, as they have the best knowledge of how their device is expected to behave when installed in a network; for example, an IP camera may need to use DNS and DHCP on the local network, and communicate with NTP servers and a specific cloud-based controller in the Internet, but nothing else. Such device behavior is manufacturer specific. Knowing each device's intended behavior allows network operators to impose a tight set of access control list (ACL) restrictions per IoT device in operation, reducing the potential attack surface on their network.

The MUD standard provides a light-weight model to enforce effective baseline security for IoT devices by allowing a network to auto-configure the required network access for the devices, so that they can perform their intended functions without having unrestricted network privileges. Many critical infrastructures and enterprises adopting IoT devices in their network and MUD white listing model can help network administrator to monitor and verify the exceptions. In addition, many standard and regulatory bodies such as NIST [11], and the European Union agency for cybersecurity [12] recommend the adoption of MUD as part of their best practices, reducing the vulnerability of

- *A. Hamza, H. Habibi Gharakheili, and V. Sivaraman are with the School of Electrical Engineering and Telecommunications, University of New South Wales, Sydney, NSW 2052, Australia (e-mails: ayyoobhamza@student.unsw.edu.au, h.habibi@unsw.edu.au, vijay@unsw.edu.au).*
- *D. Ranathunga and M. Roughan are with the ARC Centre of Excellence for Mathematical and Statistical Frontiers at the School of Mathematical Sciences, University of Adelaide, SA, 5005, Australia (e-mails: dinesha.ranathunga@adelaide.edu.au, matthew.roughan@adelaide.edu.au).*
- *T. Benson is with the School of Computer Science and Engineering, Brown University, Providence, RI 02192, USA (e-mail: tab@cs.brown.edu).*
- *This submission is an extended and improved version of our paper presented at the ACM Workshop on IoT S&P 2018 [1].*

IoT devices to botnets and other network-based threats as well as reducing the potential for harm from exploited IoT devices. MUD as a good practice to secure the IoT devices. MUD is also beneficial to manufacturers who want competitive advantages [13] since it differentiates their device by offering a network-based security feature, and hence improving customer satisfaction and/or adoption due to reduced security risks.

This paper provides solutions to improve the issues in the proposed MUD ecosystem. MUD is a new and emerging paradigm, and there is little collective wisdom today on how manufacturers should develop behavioral profiles of their IoT devices, or how organizations should use these profiles to secure their network and monitor the runtime behavior of IoT devices. Our preliminary work in [1] was one of the first attempts to address these shortcomings. This paper[1] significantly expands on our prior work by proposing an IoT device classification framework which uses observed traffic traces and incrementally compares them with known IoT MUD signatures. We use this framework and trace data captured over a period of six months from a testbed comprising of 28 distinct IoT devices to identify (a) legacy IoT devices without vendor MUD support; (b) IoT devices with outdated firmware; and (c) IoT devices which are potentially compromised. To the best of our knowledge, this is the first attempt to automatically generate MUD profiles, formally check their consistency and compatibility with an organizational policy, prior to deployment. In summary, our contributions are:

- We instrument a tool to assist IoT manufacturers to generate MUD profiles. Our tool takes as input the packet trace containing the operational behavior of an IoT device, and generates as output a MUD profile for it. We contribute our tool as open source [14], apply it to 28 consumer IoT devices, and highlight insights and challenges encountered in the process.
- We apply a formal semantic framework that not only validates a given MUD profile for consistency, but also checks its compatibility with a given organizational policy. We apply our semantic framework to representative organizations and selected devices, and demonstrate how MUD can greatly simplify the process of IoT acceptance into the organization.
- We propose an IoT device classification framework using observed traffic traces and known MUD signatures to dynamically identify IoT devices and monitor their behavioral changes in a network.

The rest of the paper is organized as follows: §2 describes relevant background work on IoT security and formal policy modeling. §3 describes our open-source tool for automatic MUD profile generation. Our verification framework for MUD policies is described in §4, followed by evaluation of results. We describe our IoT device classification framework in §5 and demonstrate its use to identify and monitor IoT behavioral changes within a network. We conclude the paper in §6.

1. This project was supported by Google Faculty Research Awards and Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS).

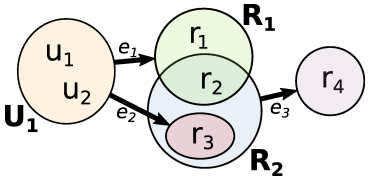


Fig. 1. A metagraph consisting of six variables, five sets and three edges.

2 Background and Related Work

Securing IoT devices has played a secondary role to innovation, *i.e.*, creating new IoT functionality (devices and services). This neglection of security has created a substantial safety and economic risks for the Internet [15]. Today many manufacturer IoT devices lack even the basic security measures [16] and network operators have poor visibility into the network activity of their connected devices, hindering the application of access-control policies to them [17]. IoT botnets continue to grow in size and sophistication and attackers are leveraging them to launch large-scale DDoS attacks [18]; devices such as baby monitors, refrigerators and smart plugs have been hacked and controlled remotely [19]; and many organizational assets such as cameras are being accessed publicly [20], [21].

Existing IoT security guidelines and recommendations [6]–[9] are largely qualitative and subject to human interpretation, and therefore unsuitable for automated and rigorous application. The IETF MUD specification [10] on the other hand defines a formal framework to capture device run-time behavior, and is therefore amenable to rigorous evaluation. IoT devices also often have a small and recognizable pattern of communication (as demonstrated in our previous work [22]). Hence, the MUD standard allows IoT device behavior to be captured succinctly, verified formally for compliance with organizational policy, and assessed at run-time for anomalous behavior that could indicate an ongoing cyber-attack.

A valid MUD profile contains a root object called "access-lists" container [10] which comprise of several access control entries (ACEs), serialized in JSON format. Access-lists are explicit in describing the direction of communication, *i.e., from-device* and *to-device*. Each ACE matches traffic on source/destination port numbers for TCP/UDP, and type and code for ICMP. The MUD specifications also distinguish *local-network* traffic from *Internet* communications.

We provide here a brief background on the formal modeling and verification framework used in this paper. We begin by noting that the lack of formal policy modeling in current network systems contribute to frequent misconfigurations [23]–[25]. We use the concept of a *metagraph*, which is a generalized graph-theoretic structure that offers rigorous formal foundations for modeling and analyzing communication-network policies in general. A metagraph is a directed graph between a collection of sets of "atomic" elements [26]. Each set is a node in the graph and each directed edge represents the relationship between two sets. Fig. 1 shows an example where a set of users (U_1) are related to sets of network resources (R_1, R_2, R_3) by the edges e_1, e_2 and e_3 describing which user u_i is allowed to access resource r_j.

Metagraphs can have attributes associated with their edges. An example is a *conditional metagraph* which includes propositions – statements that may be true or false – assigned to their edges as qualitative attributes [26]. The generating sets of these metagraphs are partitioned into a variable set and a proposition set. A conditional metagraph is formally defined as follows:

Definition 1 (Conditional Metagraph)**.** *A conditional metagraph is a metagraph* $S{=}\langle X_p \cup X_v, E\rangle$ *in which* X_p *is a set of propositions and* X_v *is a set of variables, and:*

1. at least one vertex is not null, i.e., $\forall e' \in E, V_{e'} \cup W_{e'} \neq \phi$

2. the invertex and outvertex of each edge must be disjoint, i.e., $X = X_v \cup X_p$ *with* $X_v \cap X_p = \phi$

3. an outvertex containing propositions cannot contain other elements, i.e., $\forall p \in X_p, \forall e' \in E$*, if* $p \in W_{e'}$*, then* $W_{e'} = p$*.*

Conditional metagraphs enable the specification of stateful network-policies and have several useful operators. These operators readily allow one to analyze MUD policy properties like consistency.

The MUD standard defines how a MUD profile needs to be fetched. A MUD profile is downloadable using a MUD url (*e.g.,* via DHCP). The MUD specification suggests creating a mapping of devices to their MUD urls for legacy devices already in production networks. Therefore, in this paper, we develop a method for automatic device identification using MUD profiles to reduce the complexity of mapping a device to its corresponding MUD-url manually (see §5). Our previous work [27] discussed the challenges of enforcing MUD profiles into networks. We showed how the MUD paradigm can effectively reduce the attack surface while sophisticated attacks (those conforming to MUD profiles) can still be launched on IoT devices. In other work [28], we trained machine learning-based models by network activity of MUD rules to detect volumetric attacks. The primary focus of this paper, instead, is on the pre-enforcement stage whereby network operators can use MUD profiles to ensure (prior to deployment) IoT devices are compatible with their organizational policies. Additionally, this paper develops a method to help operators identify existing devices (already deployed) in the network by progressively checking their behavior against a set of known profiles.

Past works have employed machine learning to classify IoT devices for asset management [29], [30]. Method in [29] employs over 300 attributes (packet-level and flow-level), though the most influential ones are minimum, median, and average of packet volume, Time-To-Live (TTL), the ratio of total bytes transmitted and received, and the total number of packets with RST flag reset. Work in [30] proposes to use features with less computation cost at runtime. Existing Machine learning based proposals need to re-train their model when a new device type is added – this limits the usability in terms of not being able to transfer the models across deployments.

While all the above works make important contributions, they do not leverage the MUD standard, which the IETF is pushing for vendors to adopt. We overcome the shortfall by developing an IoT device classification framework which dynamically compares the device traffic traces (run-time network behavior) with known static IoT MUD signatures. Using this framework, we are able to identify (a) legacy IoT devices without vendor MUD support; (b) IoT devices with outdated firmware; and (c) IoT devices which are potentially compromised.

TABLE 1
Flows observed for Blipcare BP (*: wildcard, proto: Protocol, sPort: source port number, dPort: destination port number).

Source	Destination	proto	sPort	dPort
*	192.168.1.1	17	*	53
192.168.1.1	*	17	53	*
*	tech.carematix.com	6	*	8777
tech.carematix.com	*	6	8777	*

3 MUD Profile Generation

The IETF MUD is a new standard. Hence, IoT device manufacturers have not yet provided MUD profiles for their devices. We therefore developed a tool – *MUDgee* – which automatically generates a MUD profile for an IoT device from its traffic trace in order to make this process faster, easier, and more accurate. Note that the generated profile completeness solely depends on the completeness of the input traffic traces. In this section, we describe the structure of our open source tool [14], apply it to traces of 28 consumer IoT devices, and highlight insights. To capture all the possible benign states, we combined both autonomous and interactive approaches. In order to autonomously capture benign behavior of IoT devices in our testbed, we installed a touch replay tool on a Samsung galaxy tab to record all possible user interactions (*e.g.,* turning on/off a lightbulb or streaming video from a camera) with individual IoTs. We also setup these devices in our lab environment and captured direct user interactions. Traffic traces were stored in an external hard disk connected to the router.

We captured traffic flows for each IoT device during a six month observation period, to generate our MUD rules. The IETF MUD standard allows both 'allow' and 'drop' rules. In our work, instead, we generate profiles that follow a whitelisting model (*i.e.,* only 'allow' rules with default 'drop'). Having a combination of 'accept' and 'drop' rules requires a notion of rule priority (*i.e.,* order) and is not supported by the IETF MUD standard. For example, Table 1 shows traffic flows observed for a Blipcare blood pressure monitor. The device only generates traffic whenever it is used. It first resolves its intended server at `tech.carematrix.com` by exchanging a DNS query/response with the default gateway (*i.e.,* the top two flows). It then uploads the measurement to its server operating on TCP port 8777 (described by the bottom two rules).

3.1 MUDgee Architecture

MUDgee implements a programmable virtual switch (vSwitch) with a header inspection engine attached and plays an input PCAP trace (of an arbitrary IoT device) into the switch. *MUDgee* has two separate modules; (a) captures and tracks all TCP/UDP flows to/from device, and (b) composes a MUD profile from the flow rules. We describe these two modules in detail below.

Capture intended flows: Consumer IoT devices use services provided by remote cloud servers and also expose services to local hosts (*e.g.,* a mobile App). We track both (intended)

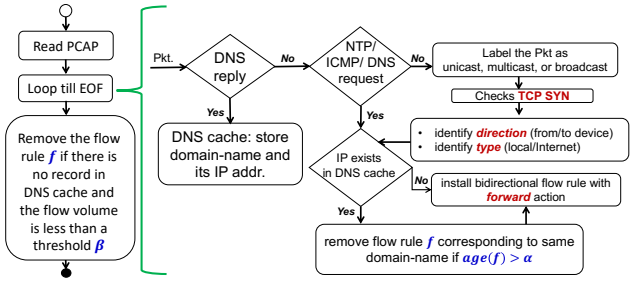


Fig. 2. Algorithm for capturing device flows and inserting reactive rules.

remote and local device communications using separate flow rules to meet the MUD specification requirements.

It is challenging to capture services (*i.e.*, especially those operating on non-standard TCP/UDP ports) that a device is either accessing or exposing. This is because local/remote services operate on static port numbers whereas source port numbers are dynamic (and chosen randomly) for different flows of the same service. We note that it is trivial to deduce the service for TCP flows by inspecting the SYN flag, but not so easy for UDP flows. We, therefore, developed an algorithm (Fig. 2) to capture bidirectional flows for an IoT device.

We first configure the vSwitch with a set of proactive rules, each with a specific action (*i.e.*, "forward" or "mirror") and a priority (detailed rules can be found in our technical report [31]). Proactive rules with a 'mirror' action will feed the header inspection engine with a copy of the matched packets. Our inspection algorithm, shown in Fig. 2, will insert a corresponding reactive rule into the vSwitch.

Our algorithm matches a DNS reply to a top priority flow and extracts and stores the domain name and its associated IP address in a DNS cache. This cache is dynamically updated upon arrival of a DNS reply matching an existing request.

The MUD specification also requires the segregation of traffic to and from a device for both local and Internet communications. Hence, our algorithm assigns a unique priority to the reactive rules associated with each of the groups: from-local, to-local, from-Internet and to-Internet. We use a specific priority for flows that contain a TCP SYN to identify if the device or the remote entity initiated the communication.

Flow translation to MUD: *MUDgee* uses the captured traffic flows to generate a MUD profile for each device. We convert each flow to a MUD ACE by considering the following:

Consideration 1: We reverse lookup the IP address of the remote endpoint and identify the associated domain name (if any), using the DNS cache.

Consideration 2: Some consumer IoTs, especially IP cameras, typically use the Session Traversal Utilities for NAT (STUN) protocol to verify that the user's mobile app can stream video directly from the camera over the Internet. If a device uses the STUN protocol over UDP, we must allow all UDP traffic to/from Internet servers because the STUN servers often require the client device to connect to different IP addresses or port numbers.

Consideration 3: We observed that several smart IP cameras communicate with many remote servers operating on the same port (*e.g.*, Belkin Wemo switch). However, no DNS responses were found corresponding to the server IP addresses. So, the device must obtain the IP address of its servers via a non-standard channel (*e.g.*, the current server may instruct the device with the IP address of the subsequent server). If a device communicates with several remote IP addresses (*i.e.*, more than our threshold value of five), all operating on the same port, we allow remote traffic to/from any IP addresses (*i.e.*, *) on that specific port number.

Consideration 4: Some devices (*e.g.*, TPLink plug) use the default gateway as the DNS resolver, and others (*e.g.*, Belkin WeMo motion) continuously ping the default gateway. The MUD standard maps local communication to fixed IP addresses through the controller construct. We consider the local gateway to act as the controller, and use the namespace `urn:ietf:params:mud:gateway` for the gateway.

Consideration 5: The MUD specification allows subnet matching for ACLs with IP endpoints, but not specifically for ACLs with domain name endpoints. There are certain devices that communicate with a large set of domain names which share the same top-level domain. For example, our instance of Chromecast fetches media contents from CDN servers with domain names such as `r4---sn-ntqe6n76.googlevideo.com` and `r3---sn-55goxu-ntqe.googlevideo.com`. It is practically infeasible to capture traffic traces that cover all domain names which can be contacted by the Chromecast, but these domain names match a single top-level domain name `*.googlevideo.com`. During the generation of MUD profiles by the MUDgee tool, such aggregation (masking) of domain names can be done by the user (network administrator) who will provide the list of preferred top-level domains. During the enforcement, having "*." in source or destination domain name fields of a MUD profile would allow any prefixes. The authors of the MUD standard may want to incorporate this amendment in future.

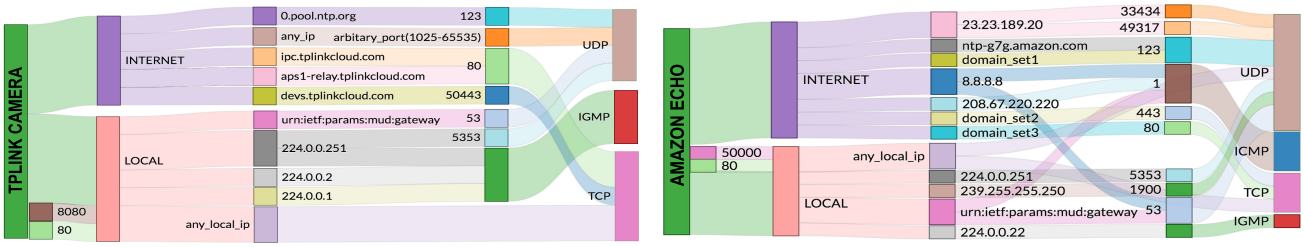
The generated MUD profiles of the 28 consumer IoT devices in our testbed are listed in Table 2 and are publicly available at: `https://iotanalytics.unsw.edu.au/mud/`.

3.2 Insights and challenges

We categorize IoT devices into three groups: (a) devices with static functionalities which can be well-defined; (b) devices with static functionalities, but cannot be completely defined due to use of dynamic IP addresses or domain names; and, (c) devices with dynamic functionalities that can be augmented by apps/recipes/redirection. In what follows, we highlight insights obtained from a representative device in each of these three categories.

(Category1) static functionality: The Blipcare BP monitor is an example of an IoT device with static functionalities. It exchanges DNS queries/responses with the local gateway and communicates with a single domain name over TCP port 8777. So, its behavior can be locked down to a limited set of static flow rules. The majority of IoT devices that we tested (*i.e.*, 22 out of 28) fall into this category (marked in green in Table 2).

(Category2) static functionality but dynamic endpoints: We use Sankey diagrams (shown in Fig. 3) to represent the MUD profiles in a human-friendly way. The second category of our generated MUD profiles is exemplified by Fig. 3(a).



(a) TP-Link camera.

(b) Amazon Echo (*e.g.*, domain_set1:0.north-america.pool.ntp.org, 1.north-america.pool.ntp.org, domain_set2: dcape-na.amazon.com, softwareupdates.amazon.com, domain_set3:kindle-time.amazon.com, live-radio01.mediahubaustralia.com, www.example.com).

Fig. 3. Sankey diagrams of MUD profiles for: (a) TP-Link camera, and (b) Amazon Echo.

TABLE 2
List of IoT devices for which we have generated MUD profiles. Devices with purely static functionality are marked in green. Devices with static functionality but loosely defined (*e.g.*, due to use of STUN protocol) are marked in blue. Devices with complex and dynamic functionality are marked in red.

Type	IoT device
Camera	Netatmo Welcome, Dropcam, Withings Smart Baby Monitor, Canary camera, TP-Link Day Night Cloud camera, August doorbell camera, Samsung SmartCam, Ring doorbell, Belkin NetCam
Air quality sensors	Awair air quality monitor, Nest smoke sensor, Netatmo weather station
Healthcare devices	Withings Smart scale, Blipcare Blood Pressure meter, Withings Aura smart sleep sensor
Switches and Triggers	iHome power plug, WeMo power switch, TPLink plug, Wemo Motion Sensor
Lightbulbs	Philips Hue lightbulb, LiFX bulb
Hub	Amazon Echo, SmartThings
Multimedia	Chromecast, Triby Speaker
Other	HP printer, Pixstar Photoframe, Hello Barbie

This Sankey diagram shows how the TP-Link camera accesses/exposes limited ports on the local network. The camera gets its DNS queries resolved, discovers local network using mDNS over UDP 5353, probes members of certain multicast groups using IGMP, and exposes two TCP ports 80 (management console) and 8080 (unicast video streaming) to local devices. All these activities can be defined by a tight set of ACLs. But, over the Internet, the camera communicates to its STUN server,accessing an arbitrary range of IP addresses and port numbers shown by the top flow. Due to this communication, the functionality of this device can only be loosely defined. Devices that fall in to this category (*i.e.*, due to the use of STUN protocol),are marked in blue in Table 2. The functionality of these devices can be more tightly defined if manufacturers of these devices configure their STUN servers to operate on a specific set of endpoints and port numbers, instead of a broad and arbitrary range.

(Category3) dynamic functionality: The Amazon Echo and Triby speaker represent devices with complex and dynamic functionalities triggered by various user interactions. Such devices (marked in red in Table 2), can communicate with a growing range of endpoints on the Internet that the original manufacturer cannot specify in advance. For example, we found that our instance of Amazon Echo communicates with "`https://meethue.com`" in response to a voice command activating the Hue lightbulb in our lab. For additional skills, however, the Amazon Echo is expected to communicate with its cloud-based backend facilitating subsequent interactions with the pertinent vendor servers. As another example, it contacted a radio streaming website "`https://ic2ti.scahw.com.au`" when the user requested a radio streaming Alexa service via the Amazon Echo mobile app. For these types of devices, the main challenge is how manufacturers dynamically update their MUD profiles, capturing their device capabilities.

The main limitation of generating a MUD profile using traffic traces is that certain flows may be missed during the packet capture, because some behaviors occur rarely (*e.g.*, firmware updates). At least we can include all possible user interactions in the traffic trace but in case of missing flows in a MUD profile, network operators are still able to find missing flows via exception packets (unconfirmed packets). Those flows can then be manually verified by the network administrator, and added to the MUD profile.

4 MUD Profile Correctness and Compliance

Network operators should not allow a device to be installed in their network, without first checking its compatibility with the organization's security policy. We've developed a tool – *MUDdy* – which can help with the task. *MUDdy* can check an IoT device's MUD profile is correct syntactically and semantically and ensure that only devices which are compliant and have MUD signatures that adhere to the IETF standard are deployed in a network.

4.1 Syntactic correctness

A MUD profile comprises of a YANG model that describes device-specific network behavior. In the current version of MUD, this model is serialized using JSON [10] and this serialization is limited to a few YANG modules (*e.g.*, ietf-access-control-list). *MUDdy* raises an invalid syntax exception when parsing a MUD profile if it detects any schema beyond these permitted YANG modules.

MUDdy also rejects MUD profiles containing IP addresses with local significance. The IETF advises MUD-profile publishers to utilize the high-level abstractions provided in the MUD standard and avoid using hardcoded private IP addresses [10]. *MUDdy* also discards MUD profiles containing access-control actions other than 'accept' or 'drop'.

4.2 Semantic correctness

Checking a MUD profile's syntax partly verifies its correctness. A profile must additionally be semantically correct; so we must check a profile, for instance, for inconsistencies.

We emphasize here that a MUD profile is an IETF standard description of permitted traffic flows for an IoT device.

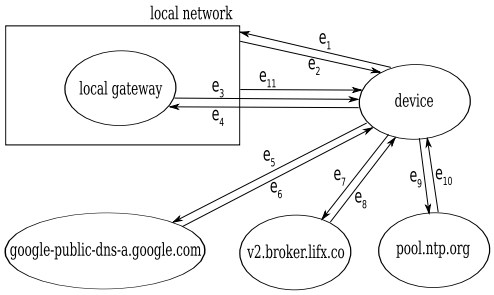


Fig. 4. Metagraph model of a LiFX bulb's MUD policy. The policy describes permitted traffic flow behavior. Each edge label has attached a set of propositions of the metagraph. For example e_4={$protocol = 17, UDP.dport = 53, UDP.sport = 0-65535, action = accept$}.

This profile contains a set of access control entries defined by MUD specification syntax along with additional details to identify the device type. Note that in order to check for semantic correctness we need to combine actual network flows with an action, therefore we relate each access control entry to a network rule and we call it a "MUD policy rule" which can be a 5-tuple, 4-tuple, or 3-tuple flow depending on how it is specified in the MUD profile.

MUD policy inconsistencies can produce unintended consequences [32] and in a MUD policy, inconsistencies can stem from (a) overlapping rules with different access-control actions; and/or (b) overlapping rules with identical actions. The MUD standard excludes rule ordering, so, the former describes ambiguous policy-author intent (*i.e.,* intent-ambiguous rules). In comparison, the latter associates a clear (single) outcome and describes *redundancies*. Our adoption of an application-whitelisting model prevents the former by design, but, redundancies are still possible and need to be checked.

MUDdy models a MUD policy using a metagraph underneath. This representation enables us to use Metagraph algebras [26] to precisely check the policy model's consistency (and hence MUD profile consistency). It's worth noting here that past works [33] classify policy consistency based on the level of policy-rule overlap. But, these classifications are only meaningful when the policy-rule order is important (*e.g.,* in a vendor-device implementation). However, rule order is not considered in the IETF MUD standard and it is also generally inapplicable in the context of a policy metagraph. Below is a summary description of the process we use to check the consistency of a policy model.

4.2.1 Policy modeling

Access-control policies are often represented using the five-tuple: source/destination address, protocol, source/destination ports [34]–[36]. We construct MUD policy metagraph models leveraging this idea. Fig. 4 shows an example for a Lifx bulb. Here, the source/destination addresses are represented by the labels `device`, `local-network`, `local-gateway` and a domain-name (*e.g.,* `pool.ntp.org`). Protocol and ports are propositions of the metagraph.

4.2.2 Policy definition and verification

We wrote *MGtoolkit* [37] – a package for implementing metagraphs – to instantiate our MUD policy models. *MGtoolkit* is implemented in Python 2.7. The API allows users to create metagraphs, apply metagraph operations and evaluate results.

Mgtoolkit provides a `ConditionalMetagraph` class which extends a `Metagraph` and supports propositions. The class inherits the members of a `Metagraph` and additionally supports methods to check consistency. We use this class to instantiate our MUD policy models and check their consistency.

Our verification of metagraph consistency uses *dominance* [26] which can be introduced constructively as follows:

Definition 2 (Edge-dominant Metapath)**.** *Given a metagraph S=$\langle X, E\rangle$ for any two sets of elements B and C in X, a metapath $M(B, C)$ is said to be edge-dominant if no proper subset of $M(B, C)$ is also a metapath from B to C.*

Definition 3 (Input-dominant Metapath)**.** *Given a metagraph S=$\langle X, E\rangle$ for any two sets of elements B and C in X, a metapath $M(B, C)$ is said to be input-dominant if there is no metapath $M'(B', C)$ such that $B' \subset B$.*

In other words, edge-dominance (input-dominance) ensures that none of the edges (elements) in the metapath are redundant. These concepts allow us to define a dominant metapath as per below. A non-dominant metapath indicates redundancy in the policy represented by the metagraph.

Definition 4 (Dominant Metapath)**.** *Given a metagraph S=$\langle X, E\rangle$ for any two sets of elements B and C in X, a metapath $M(B, C)$ is said to be dominant if it is both edge dominant and input-dominant.*

We could use the dominance property to also check for MUD policy conflicts, but since conflicts are removed by design (MUD profiles are generated using an application whitelisting model), we focus here on detecting redundancies accurately. Identifying redundancies is important because they indicate how efficiently the MUD profiles are generated using MUDgee. A high redundancy count would suggest improving the profile generation algorithms within. Our aim here is not to eliminate all redundancies because there is a trade-off between efficiency and convenience when generating MUD profiles. We want the profiles to be efficient while using the high-level abstractions provided in the MUD standard for convenience, so some level of redundancy is acceptable.

4.2.3 Compatibility with best practices

MUD policy consistency checks partly verify if it is semantically correct. In addition, a MUD policy may need to be verified against a local security policy or industry recommended practices (such as the ANSI/ISA- 62443-1-1), for compliance. Doing so, is critical when installing an IoT device in a mission-critical network such as a SCADA network, where highly restrictive cyber-security practices are required to safeguard people from serious injury or even death!

We built an example organizational security policy based on SCADA best practice guidelines to check MUD policy compliance. We chose these best practices because they offer a wide spectrum of policies representative of various organizations. For instance, they include policies for the highly protected SCADA zone (which, for instance, might run a power plant) as well as the more moderately-restrictive

Enterprise zone. Fig. 5(a) shows part of the metagraph describing this example best practice security policy with respect to permitted ICMP traffic flow behavior. So for instance, the policy permits ICMP flow between the DMZ and the Internet as well as between the DMZ and Enterprise Zone, but not between the Internet and the Enterprise Zone. This is due to ICMP's lack of built-in security to prevent a hacker from mapping or attacking a network.

We define a MUD policy rule to be SCADA (or Enterprise) zone compatible if its corresponding traffic flow complies with SCADA (or Enterprise) best practice policy. For instance, a MUD rule which permits a device to communicate with the local network using DNS complies with the Enterprise zone policy. But, a rule enabling device communication with an Internet server using HTTP violates the SCADA zone policy.

Our past work has investigated the problem of policy comparison using formal semantics, in the SCADA domain for firewall access-control policies [38]. We adapt the methods and algebras developed there, to also check MUD policies against SCADA best practices. Key steps enabling these formal comparisons are summarized below.

Policies are mapped into a unique canonical decomposition. Policy canonicalization can be represented through a mapping $c : \Phi \rightarrow \Theta$, where Φ is the policy space and Θ is the canonical space of policies. All equivalent policies of Φ map to a singleton. For $p^X, p^Y \in \Phi$, we note the following (the proof follows the definition)

Lemma 5. *Policies $p^X \equiv p^Y$ iff $c(p^X) = c(p^Y)$.*

MUD policy compliance can be checked by comparing canonical policy components. For instance

$$\text{Is } c(p^{device \rightarrow controller}) = c(p^{SCADA \rightarrow Enterprise}) \text{ ?}$$

A notation also useful in policy comparison is that policy P^A *includes* policy P^B. In SCADA networks, the notation helps evaluate whether a MUD policy is compliant with industry-recommended practices in [39], [40]. A violation increases the vulnerability of a SCADA zone to cyber attacks.

We indicate that a policy *complies* with another if it is more restrictive or included in and define the following

Definition 6 (Inclusion)**.** *A policy p^X is* included *in p^Y on A iff $p^X(s) \in \{p^Y(s), \phi\}$,* i.e., *$X$ either has the same effect as Y on s, or denies s, for all $s \in A$. We denote inclusion by $p^X \subset p^Y$.*

A MUD policy (MP) can be checked against a SCADA best practice policy (RP) for compliance using inclusion

$$\text{Is } p^{MP} \subset p^{RP} \text{ ?}$$

The approach can also be used to check if a MUD policy complies with local security policies of an organization, ensuring that IoT devices are plug-and-play enabled only in the compatible zones of the network. For instance, a network operator may wish to install an IoT device (*e.g.,* an Amazon Echo) in the Enterprise Zone for easier real-time weather and traffic updates. Verifying that the device's MUD policy complies with the organizational security policy prior to installation is necessary. Fig. 5(b) shows the MUD policy of the Amazon Echo, superimposed on the organizational best practice policy, describing permitted vs actual device ICMP flow. An inclusion check of the MUD policy against the best practice policy (which involves canonical decomposition of the policies) informs us of a policy conflict. In this case, the device's MUD policy fails compliance (due to the device's use of ICMP to communicate on the Internet), informing the network operator that it should not be installed in the Enterprise Zone, and why.

TABLE 3
MUD policy analysis summary for our testbed IoT devices using *Muddy* (*Safe to install?* indicates where in a network (*e.g.,* Enterprise Zone, SCADA Zone, DMZ) the device can be installed without violating best practices, DMZ - Demilitarized Zone, Corp Zone - Enteprise Zone). *Muddy* ran on a standard desktop computer; *e.g.,* Intel Core CPU 2.7-GHz computer with 8GB of RAM running Mac OS X)

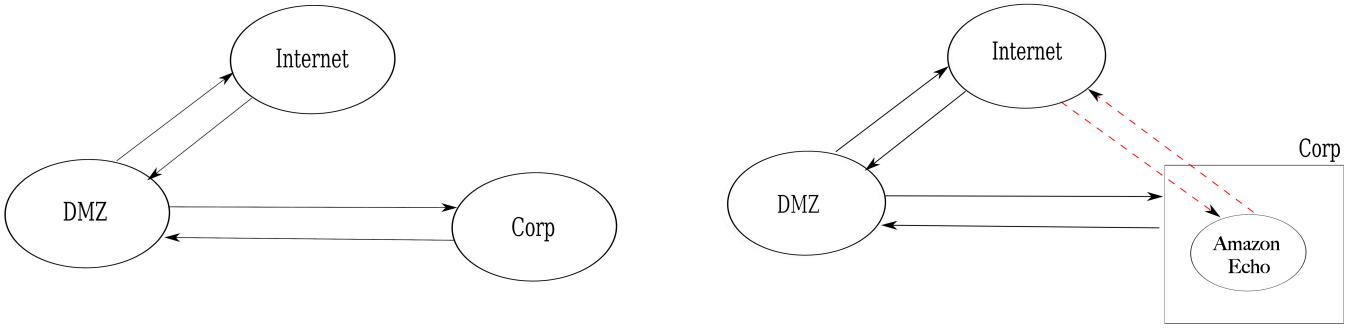
Device name	#MUD profile rules	#Redundant rules	Redundancy check time (s)	Compliance check time (s)	Safe to install ?	% Rules violating SCADA	% Rules violating Corp
Blipcare bp	6	0	0.06	38	DMZ, Corp Zone	50	0
Netatmo weather	6	0	0.04	36	DMZ, Corp Zone	50	0
SmartThings hub	10	0	1	39	DMZ, Corp Zone	60	0
Hello barbie doll	12	0	0.6	38	DMZ, Corp Zone	33	0
Withings scale	15	4	0.5	40	DMZ, Corp Zone	33	0
Lifx bulb	15	0	0.8	42	DMZ, Corp Zone	60	0
Ring door bell	16	0	1	39	DMZ, Corp Zone	38	0
Awair air monitor	16	0	0.3	101	DMZ, Corp Zone	50	0
Withings baby	18	0	0.2	41	DMZ, Corp Zone	28	0
iHome power plug	17	0	0.1	42	DMZ	41	6
TPlink camera	22	0	0.4	40	DMZ	50	4
TPlink plug	25	0	0.6	173	DMZ	24	4
Canary camera	26	0	0.4	61	DMZ	27	4
Withings sensor	28	0	0.2	71	DMZ	29	4
Drop camera	28	0	0.3	214	DMZ	43	11
Nest smoke sensor	32	0	0.3	81	DMZ	25	3
Hue bulb	33	0	2	195	DMZ	27	3
Wemo motion	35	0	0.4	47	DMZ	54	8
Triby speaker	38	0	1.5	187	DMZ	29	3
Netatmo camera	40	1	0.9	36	DMZ	28	2
Belkin camera	46	3	0.9	55	DMZ	52	11
Pixstar photo frame	46	0	0.9	43	DMZ	48	28
August door camera	55	9	0.8	38	DMZ	42	13
Samsung camera	62	0	1.7	193	DMZ	39	19
Amazon echo	66	4	3.2	174	DMZ	29	2
HP printer	67	10	1.8	87	DMZ	25	9
Wemo switch	98	3	3.1	205	DMZ	24	6
Chrome cast	150	24	1.1	56	DMZ	11	2

4.3 Correctness and Compatibility Results

We ran *MUDgee* on a standard laptop computer (*e.g.,* Intel Core CPU 3.1 GHz computer with 16GB of RAM running Mac OS X) and generated MUD profiles for 28 consumer IoT devices installed in our testbed. *MUDgee* generated these profiles by parsing a 2.75 Gb PCAP file (containing 4.5 months of packet trace data from our testbed), within 8.5 minutes averaged per device. Table 3 shows a high-level summary of these MUD profiles.

It should be noted that a MUD profile generated from a device's traffic trace can be incorrect if the device is compromised, as the trace might include malicious flows. In addition, the generated MUD profile is limited to the input trace. Our tool can be extended by an API that allows manufacturers to add rules that are not captured in the PCAP trace.

Zigbee, Z-wave and bluetooth technologies are also increasingly being used by IoT devices. Thus, such devices come with a hub capable of communicating with the Internet. In such cases, a MUD profile can be generated only for the hub.



(a) Example Best Practice Security Policy describing permitted ICMP traffic flow behavior.

(b) Amazon Echo's MUD policy superimposed on Fig. 5(a) describing its ICMP traffic flow behavior.

Fig. 5. Metagraph models of policies for: (a) Organizational Best Practice Security Policy for permitted ICMP traffic behavior between three zones: Internet, DMZ and Enterprise Zone, and (b) Amazon Echo's MUD policy superimposed on part (a) describing its ICMP flow behavior. We assume here an Enterprise Zone installation of the device.

We then ran *MUDdy* on a standard desktop computer (*e.g.,* Intel Core CPU 2.7-GHz computer with 8GB of RAM running Mac OS X) to automatically parse the generated MUD profiles and identify inconsistencies within them. Our adoption of an application whitelisting model restricts inconsistencies to redundancies. We determined non-dominant metapaths (as per Definition 4) in each policy metagraph built by *MUDdy*, to detect redundancies. The average times (in milliseconds) taken to find these redundancies are shown in Table 3.

As the table shows, there were for instance, three redundant rules present in the Belkin camera's MUD policy. These rules enabled ICMP traffic to the device from the local network as well as the local controller, making the policy inefficient.

Table 3 also illustrates the results from our MUD policy best-practice compliance checks. For instance, a Blipcare blood pressure monitor can be safely installed in the Demilitarized zone (DMZ) or the Enterprise zone but not in a SCADA zone: 50% of its MUD rules violate the best practices, exposing the zone to potential cyber-attacks. Policy rules enabling the device to communicate with the Internet directly, trigger these violations.

In comparison, an Amazon echo speaker can only be safely installed in a DMZ. Table 3 shows that 29% of the device's MUD rules violate the best practices if it's installed in the SCADA zone. Only 2% of the rules violate if it's installed in the Enterprise zone. The former violation stems from rules which for instance, enable HTTP to the device. The latter is due to rules enabling ICMP to the device from the Internet.

MUDdy's ability to pinpoint to MUD rules which fail compliance, helps us to identify possible workarounds to overcome the failures. For instance, in the Belkin camera, local DNS servers and Web servers can be employed to localize the device's DNS and Web communications to achieve compliance in the SCADA zone.

4.4 MUD recommendations

At present, the MUD specification allows both accept and drop rules but does not specify priority, allowing ambiguity. This ambiguity is removed if only accept rules (*i.e.,* whitelisting) is used. Whitelisting means metagraph edges describe enabled traffic flows. So, the absence of an edge implies two metagraph nodes don't communicate with one another. But when drop rules are introduced, an edge also describes prohibited traffic flows, hindering easy visualization and understanding of the policy. We recommend the MUD standard be revised to only support explicit 'accept' rules.

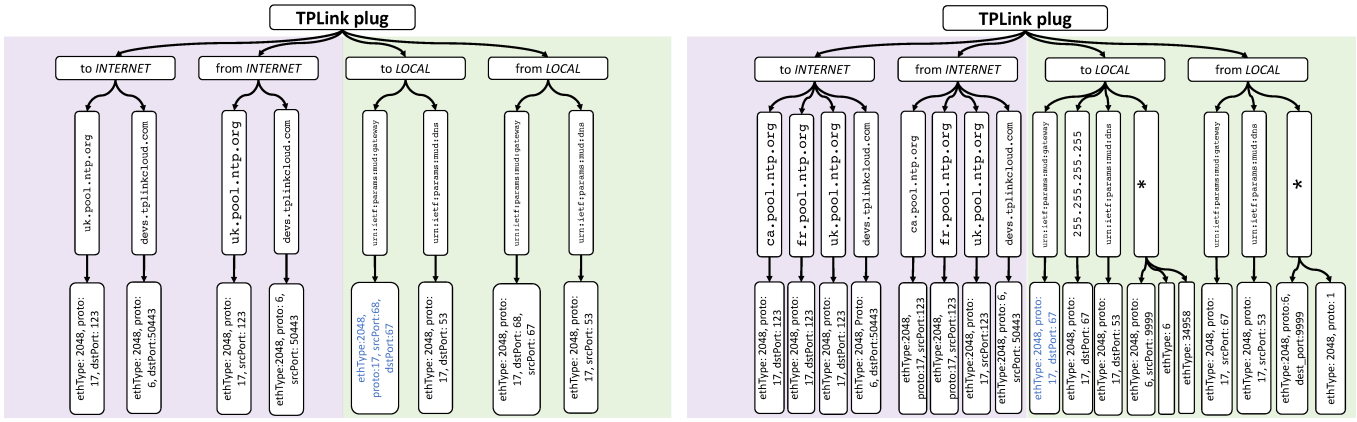
The MUD standard also does not support private IP addresses, instead profiles are made readily transferrable between networks via support for high-level abstractions. For instance, to communicate with other IoT devices in the network, abstractions such as *same-manufacturer* is provided.

The MUD standard however, permits the use of public IP addresses. This relaxation of the rule allows close coupling of policy with network implementation, increasing its sensitivity to network changes. A MUD policy describes IoT device behavior and should only change when its actual behavior alters and not when network implementation changes! Hardcoded public IP addresses can also lead to accidental DoS of target hosts. A good example is the DoS of NTP servers at the University of Wisconsin due to hard-coded IP addresses in Netgear routers [41]. We recommend that support for explicit public IP addresses be dropped from the MUD standard.

5 Checking Run-Time Profile of IoT Devices

In this section, we describe how the network behaviors of IoT devices are tracked at run-time, mapping the behavior of each device to one of a set of known MUD profiles. This is needed for managing legacy IoTs that do not have support for the MUD standard. A MUD profile is a simple and environment-neutral description of IoT communications, and hence allows us to develop a simple model to identify corresponding devices. To do so, a behavioral profile is automatically generated and updated at run-time (in the form of a tree) for an IoT device, and a quantitative measure of its "similarity" to each of the known static MUD profiles (*e.g.,* provided by manufacturers) is calculated. It is noted that computing similarity between two such profiles is a non-trivial task.

Profile structure: A device profile has two main components, namely "Internet" and "Local" communication channels, as shown by purple and green areas in Fig. 6. Each profile is organized into a tree-like structure containing a set of nodes with categorical attributes (*i.e.,* end-point, protocol, port number over Internet/Local channels) connected



(a) 30-minutes of traffic capture.

(b) 480-minutes of traffic capture.

Fig. 6. Run-time profile of a TPLink power plug generated at two snapshots in time: (i) after 30 minutes of traffic capture; and (ii) after 8 hours of traffic capture. As observable the profile grows over time by accumulating nodes and edges.

through edges. Following the root node in each tree, there are nodes representing the channel/direction of communication, endpoints with which the device communicates, and the flow characteristics (*i.e.,* the leaf node). The run-time profile of a device (given a set of known MUD profiles) is generated using a method similar to that described in §3, with minor modifications, as described below.

The *MUDgee* tool tracks the traffic volume exchanged in each direction of UDP flows distinguishing the server and the client. However, this would lead to a high consumption of memory for generating run-time profiles. Therefore, given a UDP flow, all known MUD profiles are searched for an overlapping region on either the IoT side or the remote side. If an overlapping region is found, then the tree structure is updated with intersecting port ranges – this can be seen in Fig. 6 where the leaf node shown in light-blue text has been changed according to known MUD profiles. If no overlap is found with the MUD profiles, then the UDP flow is split into two leaf nodes: two flows matching the UDP source port (with a wild-carded destination) and the UDP destination port (with a wild-carded source) separately. This helps to identify the server side by a subsequent packet matching either of these two flows.

Metrics: We denote the run-time and MUD profile of an IoT type (i) by sets R and M_i, respectively. Each element of these two sets is represented by a branch of the tree structure shown in Fig. 6. The run-time profile R is progressively developed over time based on the traffic observed on the network, and it grows until the complete behavior of the device is captured. For a given IoT device, the similarity of its R with a number of known M_i's is calculated.

There are a number of metrics for measuring the similarity of two sets. For example, the *Jaccard index* has been widely used for comparing two sets of categorical values, and defined by the ratio of the size of the intersection of two sets to the size of their union, *i.e.,* $|R \cap Mi|/|R \cup Mi|$. Inspired by the Jaccard index, we define the following two metrics:

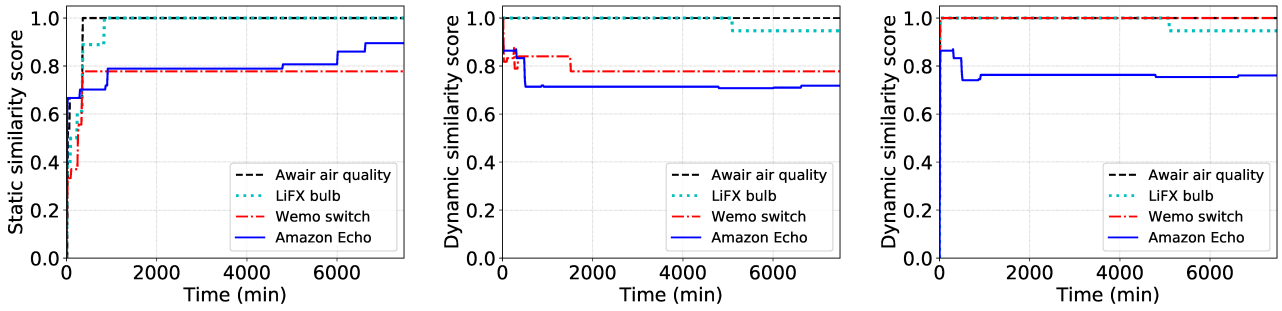
- **Dynamic similarity score:** $sim_d(R, M_i) = \frac{|R \cap M_i|}{|R|}$
- **Static similarity score:** $sim_s(R, M_i) = \frac{|R \cap M_i|}{|M_i|}$

These two metrics collectively represent the Jaccard index, each reflecting the degree of similarity from the viewpoint of either run-time or MUD profile. Note that the Jaccard index gives a combined similarity, and hence is unable to indicate the cause of variation between the two sets (*i.e.,* R and M_i). Having two fine-grained metrics enables our scheme to gain a richer visibility into similarity and achieve a faster convergence in identifying IoT devices with high accuracy (explained in §5.1 and §5.2). Each metric can take a value between 0 (*i.e.,* dissimilar) and 1 (*i.e.,* identical).

Similarity scores can be computed either periodically (every epoch) or triggered by an event (when a change is observed in the profile). In the periodic approach, increasing epoch time would cause delay in the identification process, while reducing the epoch time would lead to a higher computation cost which is unnecessary especially when run-time profiles update slowly. On the other hand, the event-based approach may seem appealing, but can also be computationally challenging especially when device behaviors are fairly dynamic (*e.g.,* IP cameras communicate with their STUN server, verifying remote users streaming video [1]). Network administrators can take either of these approaches based on their requirements and available compute resources. In this paper, we choose to compute similarity scores periodically, every 15 minutes (our epoch time). When computing $|R \cap Mi|$, redundant branches of the run-time profile are temporarily removed based on the MUD profile that it is being checked against. This assures that duplicate elements are pruned from R when checking against each M_i.

The run-time profile grows over time by accumulating nodes (and edges), as shown in Fig. 6, for example. It is seen in Fig. 6(a) that the run-time profile of a TP-Link power plug consists of 8 elements (*i.e.,* edges), 30 minutes after commencement of this profile generation. As shown in Fig. 6(b), the element count of the profile reaches 15 when more traffic an additional 450 minutes) of the device is considered.

At the end of each epoch, a device (or a group of devices) will be chosen as the "winner" that has the maximum similarity score with the IoT device whose run-time profile is being checked. It is expected to have a group of winner devices when the dynamic similarity is considered, especially when only a small subset of device behavioral profile is observed – the number of winners will reduce as the run-



(a) Static similarity score. (b) Dynamic similarity score. (c) Dynamic similarity score (SSDP excluded).

Fig. 7. Time-trace of dynamic and static similarity scores for the winners of four IoT devices. Convergence time depends on the behavior complexity of the device; for example, the static similarity score of the LiFX bulb converges to 1 within 1000 minutes whereas it takes about 12 days for the more complex Amazon echo to converge.

time profile grows over time.

Fig. 7 shows graphs of the winner similarity scores as a function of time for selected IoT devices, including the Awair air quality sensor, the LiFX bulb, the WeMo switch, and the Amazon Echo. In these plots, the winner is correctly identified for all of these four IoTs. Fig. 7(a) shows that the static similarity score grows slowly over time, and in a non-decreasing fashion. The convergence time depends on the complexity of the device behavioral profile. For example, the static similarity of the Awair air quality and LiFX bulb devices converges to 1 (*i.e.,* full score) within 1000 minutes. But for the Amazon Echo, it takes more time to gradually discover all flows, ultimately converging to the full score in about 12 days.

Also, there are IoT devices for which the static similarity might never converge to 1. For example, the WeMo switch and WeMo motion devices use a list of hard-coded IP addresses (instead of domain names) for their NTP communications. These IP addresses, however, do not serve the NTP service anymore, and consequently no NTP reply flow is captured. Similarly, it was observed that the TPLink plug uses the "`s1b.time.edu.cn`" address for NTP communications, and this domain name also seems to be not operational anymore. In addition, devices such as the August doorbell and Dropcam contact public DNS resolvers (*e.g.,* `8.8.4.4`) if the local gateway fails to respond to a DNS query of the IoT device, meaning that this specific flow will only be captured if there is an Internet outage.

On the other hand, in Fig. 7(b) the dynamic similarity score grows quickly (it may even reach a value of 1, meaning $R \subset M_i$). It may stay at 1 if no variation is observed – variation is the complement of the dynamic similarity measured in the range of $[0, 1]$ and computed as $1 - sim_d$. The Awair air quality sensor is an example of such behavior, as shown by dashed black lines in Fig. 7(b) – 19 out of 28 IoT devices in the testbed were found to behave similarly to the Awair air quality sensor in their dynamic similarity score. In some other cases, this score may slightly fall and rise again. Note that a fluctuating dynamic similarity never meets 1 due to missing elements (*i.e.,* variations). Missing elements can arise for various reasons, including: (a) MUD profile is unknown or not well-defined by the manufacturer, (b) the device firmware is old and not up-to-date, and (c) the IoT device is compromised or under attack.

During testing, we found that 9 of our lab IoTs had slight

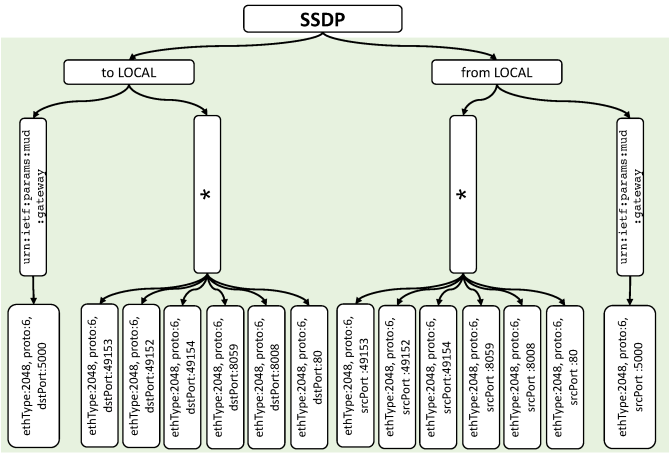


Fig. 8. SSDP runtime profile describing all discovery communications across all devices in the network.

variations for two reasons: firstly, responding to discovery requests in Local communications, if they support SSDP protocol[2] – these responses cannot be tightly specified by the manufacturer in the MUD profile since such flows depend on the environment in which the IoT device is deployed. The WeMo switch is an example of this group, as shown by dashed-dotted lines in Fig. 7(b). To address this issue, all discovery communications were used to generate a separate profile (shown in Fig. 8) by inspecting SSDP packets exchanged over the Local network. The SSDP server port number on the device can change dynamically, thus the inspection of the first packet in a new SSDP flow is required. The second reason is that missing DNS packets leads to the emergence of a branch in the profile with an IP address as the end-point instead of a domain name. This rarely occurs in our testbed network, because every midnight the process starts storing traffic traces into a new PCAP file, and thus a few packets can be lost during this transition to a new PCAP file. Missing a DNS packet was observed for the LiFX bulb, as shown by dotted lines in Fig. 7(b).

In view of the above, SSDP activity is excluded from local communications of IoT devices to obtain a clear runtime profile. As shown in Fig. 7(c) , without SSDP activity, the dynamic similarity score is able to correctly identify the correct winner for the WeMo switch within a very short time

2. A device which supports Simple Service discovery protocol advertises its capabilities to Multicast UDP port 1900. Typically the payload contains device information including IP address, name, UUID, management URL, functionalities.

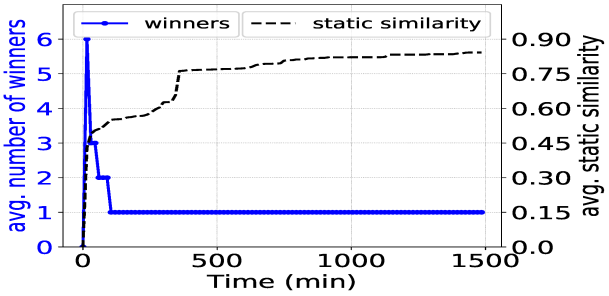


Fig. 9. Time trace of winners count and static similarity score averaged across 27 testbed IoT devices. The former shows six winners on average at the beginning of the identification process. This count drops to a single winner in less than three hours. Even with a single winner, the static similarity needs about ten hours on average to exceed a threshold of 0.8.

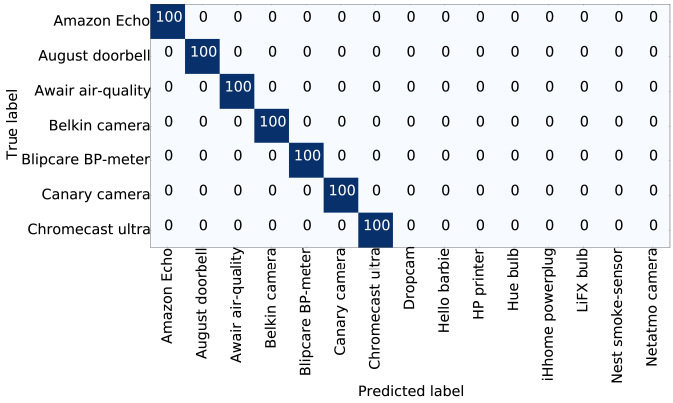


Fig. 10. Partial confusion matrix of true vs predicted device labels. The cell values are in percentages. As the table shows, for instance, the Amazon Echo (first row) is always predicted as the sole winner in all epochs. Hence, a value of 100% is recorded in the first column and 0% in the remaining columns.

interval.

Lastly, it is important to note that similarity scores (both static and dynamic) can be computed at an aggregate level (*i.e.,* combining Local and Internet channels), or for individual channels, meaning one score for the Local and one for the Internet channel. The two scores might not converge in some cases where the Local channel similarity chooses a winner while the Internet channel similarity finds a different winner device. Per-channel similarity never results in a wrong winner, though it may result in no winner. However, the aggregate similarity may end up identifying an incorrect winner, especially when the Local activity becomes dominant in the behavioral profile. This is because many IoTs have a significant profile overlap in their Local communications (*e.g.,* DHCP, ARP, or SSDP). Therefore, the per-channel similarity is checked first. If the two channels disagree, the process switches to aggregate similarity to identify the winner.

In what follows, we explain how the profile structure and the metrics are used for identifying IoT devices. In §5.1, we develop our identification process based on MUD profiles, and demonstrate its efficacy in an ideal scenario. In §5.2, we enhance our scheme to overcome practical challenges such as (a) expected MUD profile is unknown (legacy IoT devices without vendor MUD support), (b) IoT devices with outdated firmware, and (c) IoT devices that are potentially compromised or under attack, that can arise in real environments.

5.1 Identifying IoT Devices at Run-Time

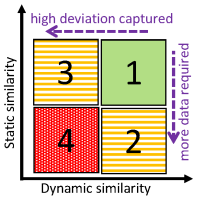
Dataset: We use packet traces (*i.e.,* `PCAP` files) collected from our testbed including a gateway (*i.e.,* a TP-Link Archer C7 flashed with the OpenWrt firmware) that serves a number of IoT devices. We use `tcpdump` tool to capture and store all network traffic (Local and Internet) onto a 1TB USB storage connected to this gateway. The resulting traffic traces span three months, starting from May 2018, containing traffic corresponding to devices listed in Table 2 (excluding Withings baby monitor). The *MUDgee* was used to generate the MUD profiles for the IoT devices in the testbed. We also developed an application over our native SDN simulator [42] to implement our identification process. We considered a smart home setting for our experiments – it is envisaged that a cloud-based security service is provided (*e.g.,* by ISP) to secure smart home devices. In our previous work [27], we demonstrated how MUD rules are automatically enforced into off-the-shelf home gateways using SDN techniques without user intervention. A similar method can be applied to enterprise networks where devices communicate with servers, on-premise or on the cloud. Note that IoT devices such as IP cameras, motion sensors, and bulbs that we used in our experiments are also found in enterprise networks.

Identification Process: As explained above, the dynamic similarity score converges faster than the static similarity score. The device identification process begins by tracking dynamic similarity at the channel level, and continues as long as the channels still agree (*i.e.,* they both choose the same winner). Depending on the diversity of observed traffic to/from the IoT device (Local versus Internet), there can be multiple winners at the beginning of the process. In this case, the static similarity is fairly low, since a small fraction of the expected profile is likely to be captured in a short time interval. This means that the process needs to see additional traffic from the device before it concludes.

Fig. 9 shows the time evolution of the winners count and static similarity, averaged across all 27 IoT devices in the testbed. Focusing on the solid blue line (left y-axis), there were up to 6 winners on average at the beginning of the identification process. The winners count gradually comes down (in less than three hours) to a single winner, and stabilizes. Even with a single winner, the static similarity, shown by dashed black lines (right y-axis), needs about ten hours on average to exceed a score of 0.8.

Note that the similarity may take a very long time to reach the full score of 1 (sometimes, it may never reach the full score as explained earlier in Fig. 7). Therefore, a complete capture of MUD flows is not guaranteed. It is up to the operator to choose an appropriate threshold at which this process concludes – a higher threshold increases the confidence level of the device identification, but it comes at the cost of longer convergence time. Thus the dynamic similarity (starting with channel level similarity, and possibly switching to aggregate level) is used to identify the winner IoT at run-time. The static similarity, on the other hand, is used to track the confidence level – an indication of safe convergence if the dynamic similarity of full score is not reached.

To evaluate the efficacy of IoT device identification at run-time, the traces collected in 2018 (*i.e.,* Data-2018) were



State	Dynamic similarity	Static similarity	Correctly Identified?	More data Required?	Deviation Captured?
1	High	High	Yes		
2	High	Low	?	Yes	
3	Low	High	Yes		Yes
4	Low	Low	?	Yes	Yes

Fig. 11. Plot of dynamic similarity vs static similarity depicting 4 distinct states. In state-1, both dynamic and static similarity scores are high and we obtain a single correct winner. In state-2, dynamic similarity is high but static similarity is low (usually occurs when only a small amount of traffic is observed). State-3 describes a region with high static similarity yet low dynamic similarity, indicating high-deviation at run time (*e.g.*, due to old firmware or device being compromised). In state-4 both dynamic and static similarity scores are low indicating a significant difference between the run-time and MUD profiles.

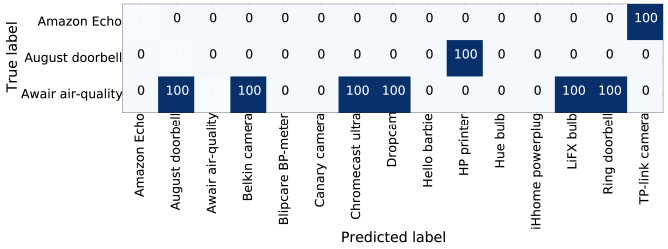


Fig. 12. Partial confusion matrix for when the intended MUD profile is absent for each device being checked.

replayed into the packet simulator tool. Fig. 10 is a confusion matrix of the results, where the rows are true labels, the columns are the predicted labels, and the cell values are percentages. For example, the first row shows that the Amazon Echo is always predicted as the sole winner in each and every epoch of the identification process, thus 100% in the first column and 0% in the remaining columns – no other device is identified as the winner in any single epoch time.

Looking at the Dropcam row, it is identified as multiple devices (*i.e.*, more than one winner) for some epochs – non-zero values are seen against all columns. But, it is important to note that Dropcam is always one of the winners, thus 100% against the Dropcam column. Further, it is also identified for example as the Amazon Echo in 0.4% of epochs.

A 100% correct convergence was observed for all devices except for the Netatmo camera, whereby it is not correctly identified in 2.3% of epochs. This mis-identification occurs due to missing DNS packets where some flows were incorrectly matched on STUN related flows (with wild-carded endpoints) of the Samsung camera and the TP-Link camera. However, this mis-identification occurred only during the first few epochs and then it converged to the correct winner.

In what follows, we discuss changes in IoT traffic behavior in the network.

5.2 Monitoring Behavioral Change of IoTs

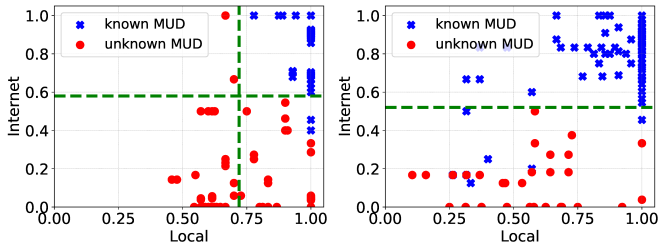
In a real environment, there are several challenges to correctly identify an IoT device at run-time: (a) there might be a device on the network for which no MUD profile is known, (b) the device firmware might not be up-to-date (thus, the run-time profile would deviate from its intended known MUD profile), and/or (c) the device might be under attack or even fully compromised. Each of these three challenges and their impact on the similarity score (both dynamic and static) are discussed below.

Fig. 11 depicts a simplified scatter plot of dynamic similarity versus static similarity, highlighting how these two metrics are interpreted. On the plot, states are labeled as 1, 2, 3, and 4. The ideal region is the quadrant highlighted for state-1 whereby both dynamic and static scores are high, and there is a single and correctly identified winner. Considering state-2 in this figure, there is a high score of dynamic similarity, whereas the static similarity is fairly low. This score combination is typically expected when a small amount of traffic from the device is observed, and more traffic is needed to determine whether the dynamic similarity continues to maintain a high score and the static similarity possibly starts rising. In state-3, having a low dynamic similarity is alarming, given the high score in the static similarity – indicating high variations at run-time. This score combination is observed when many flows observed in the device traffic are not listed in the intended MUD profile for two possible reasons: (a) the device firmware is not current, or (b) the device is under attack (or even compromised). Lastly, having low scores in both dynamic and static similarity metrics highlights a significant difference (or small overlap) between the run-time and MUD profiles. This scenario likely results in identification of an incorrect winner.

To summarize, IoT network operators may need to set threshold values for both dynamic and static similarity scores to select the winner device. Also, the identification process needs to begin with the channel-level similarity (for both dynamic and static metrics) avoiding a biased interpretation, and may switch to aggregate-level in the absence of convergence. The impact of three scenarios impacting the IoT behavioral changes is described below.

MUD profile unknown: To investigate this scenario, the MUD profile of each device was removed from the list of known MUDs. Fig. 12 shows the partial results for selected devices. Unsurprisingly, devices on the rows are identified as others (*i.e.*, one or multiple wrong winners selected), since their intended MUD profile is not present when checked at run-time. For example, the Amazon Echo converges to identification as a TP-Link camera, and the Awair air quality sensor is consistently identified as six other IoT devices. Ideally, there should not be any one device identified as the winner. Note that these results are obtained while no thresholding is applied to the similarity scores, and only the maximum score indicates the winner.

Fig. 13 shows scatter plots of channel-level scores for dynamic and static similarity metrics, respectively. The 2018 dataset was used to generate two sets of results: one with MUD profiles of the devices (shown by blue cross markers), and the other without their MUD profiles (shown by red circle markers), across all 27 IoT devices. For the dynamic similarity in Fig. 13(a), having two thresholds (*i.e.*, about 0.60 on the Internet channel and 0.75 on the Local channel) would filter incorrect instances. For the static similarity in Fig. 13(b), a threshold of 0.50 on the Internet channel is sufficient to avoid incorrect identifications. This single threshold is because the IoT profile on the Internet channel varies significantly for consumer devices (in the testbed setup), but enterprise IoTs may tend to be active on the Local network – thus a different thresholding is generally



(a) Dynamic similarity score. (b) Static similarity score.

Fig. 13. Scatter plots of channel-level scores for dynamic and static similarity metrics across 27 testbed IoT devices. Each plot depicts two sets of results: one for known MUD (blue markers) and the other for unknown MUD (red markers). Enforcing two thresholds (*i.e.*, about 0.60 on the Internet channel and 0.75 on the Local channel) would filter incorrect matches found using dynamic similarity. A threshold of 0.50 on the Internet channel is sufficient to avoid false identification when using static similarity.

TABLE 4
Identification results for data 2016.

IoT device	Profile change	Convergence with threshold				Endpoint compacted		
		Known MUD			Unknown MUD	Known MUD		Unknown MUD
		Correctly identified (%)	Incorrectly identified (%)	State	Incorrectly identified (%)	Correctly identified (%)	Incorrectly identified (%)	Incorrectly identified (%)
Amazon Echo	Yes	65.7	0	3	0	65.7	0	0
August doorbell	Yes	0	0	4	0	100	0	0
Awair air quality	Yes	100	0	1	0	100	0	0
Belkin camera	Yes	100	0	1	0	100	0	0
Blipcare BP meter	No	100	0	1	0	100	0	0
Canary camera	No	100	0	1	0	100	0	0
Dropcam	Yes	95.9	0	3	0	100	0	0
Hello barbie	No	100	0	1	0	100	0	0
HP printer	Yes	3.6	0	4	0	99.8	0	0
Hue bulb	Yes	0	0	4	0	90.6	0	0
iHome power plug	Yes	0.5	0	4	0	100	0	0
LiFX bulb	No	100	0	1	5.3	100	0	5.3
Nest smoke sensor	Yes	0	0	4	0	100	0	0
Netatmo camera	Yes	97.3	0	3	0	99	0	0
Netatmo weather	No	100	0	1	0	100	0	0
Pixstar photoframe	No	100	0	1	0	100	0	0
Ring doorbell	Yes	99.6	0	3	0	97.9	0	0
Samsung smartcam	Yes	97.6	0	1	0	97.6	0	0
Smart Things	No	100	0	1	0	100	0	0
TPlink camera	Yes	100	0	3	0	100	0	0.9
TPlink plug	Yes	100	0	1	0	100	0	0
Triby speaker	Yes	39.9	0	3	0	99.8	0	0
WeMo motion	No	100	0	1	0.7	100	0	27.3
WeMo switch	Yes	0	100	1	100	0	100	100

required for each network.

It is important to note that a high threshold would increase the identification time, and a low threshold accelerates the process but may lead to identification of a wrong winner. It is therefore up to the network operator to set appropriate threshold values. One conservative approach would be to accept no variation in the dynamic similarity, requiring a full score of 1 along with a static similarity score of more than 0.50 for each of the Local and Internet channels. For example, the results were regenerated by setting conservative thresholds mentioned above, and thus no winner was identified due to low scores in both dynamic and static similarity metrics, as shown by the state-4 quadrant in Fig. 11. This indicates that IoT devices, in absence of their MUD profiles, are consistently found in state-4, flagging possible issues.

Old firmware: IoT devices either upgrade their firmware automatically by directly communicating with a cloud server, or may require the user to confirm the upgrade (*e.g.*, the WeMo switch) via an App. For the latter, devices will remain behind the latest firmware until the user manually updates them. To illustrate the impact of old firmware, packet traces collected from the testbed over a duration of six months starting in October 2016 were used to generate run-time profiles against MUD profiles generated from data 2018. Table 4 below shows the results from data 2016.

The column labeled "Profile changed" indicates whether any changes on device behavior were observed (*i.e.*, verified

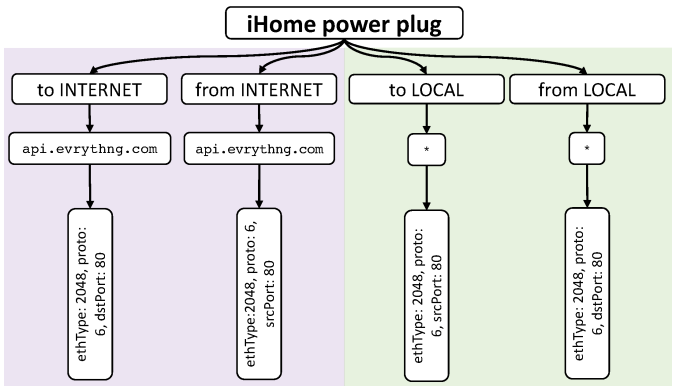


Fig. 14. Tree structure depicting profile difference (*i.e.*, R - M) for the iHome power plug.

manually) from the data 2016 dataset, compared to data 2018. These behavioral changes include endpoints and/or port number. For example, the TP-Link camera communicates with a server endpoint "`devs.tplinkcloud.com`" on TCP 50443 according to the data 2016. However, this camera communicates with the same endpoint on TCP 443 in the data 2018. Additionally, in the data 2018 dataset, an endpoint "`ipcserv.tplinkcloud.com`" is observed, which did not exist in the data 2016.

The "Convergence" column in Table 4 shows the performance of the device identification process (converging to a single winner) without thresholding, for two scenarios, namely known (*i.e.*, with) MUD and unknown (*i.e.*, without) MUD. When MUD profiles of device are known (*i.e.*, present), all devices except the WeMo switch converge to the correct winner. Surprisingly, the WeMo switch is consistently identified as the WeMo motion – even the static similarity increases to 0.96. This is because both WeMo motion and WeMo switch share the same cloud-based endpoint for their Internet communications in data 2016, but these endpoints have changed for the WeMo switch (but not for the WeMo motion) in data 2018. It is important to note that the primary objective is to secure IoT devices by enforcing tight access-control rules to network elements. Therefore, the WeMo switch can be protected by the rules of the WeMo motion until it is updated to the latest firmware. Once the WeMo switch is updated, the intrusion detection process may generate false alarms, indicating the need for re-identification.

As discussed above, a threshold is required to improve the identification process, discovering unknown devices or problematic states. Therefore, thresholds determined using the data 2018 were applied and the results are shown in the column labeled as "Convergence with threshold" in Table 4. Devices that did not have behavioural changes (from 2016 to 2018), converge correctly and appear in perfect state-1. Looking into other devices, for example the Amazon Echo, only 65.7% of instances are correctly identified – it took a while for the identification process to meet the expected thresholds set for similarity scores.

It is observed that devices with profile changes are found in state-3 or state-4. In order to better understand the reason for a low score in dynamic similarity, the profile difference can be visualized in the form of a tree structure. For example, this difference (*i.e.*, $R - M$) is shown in

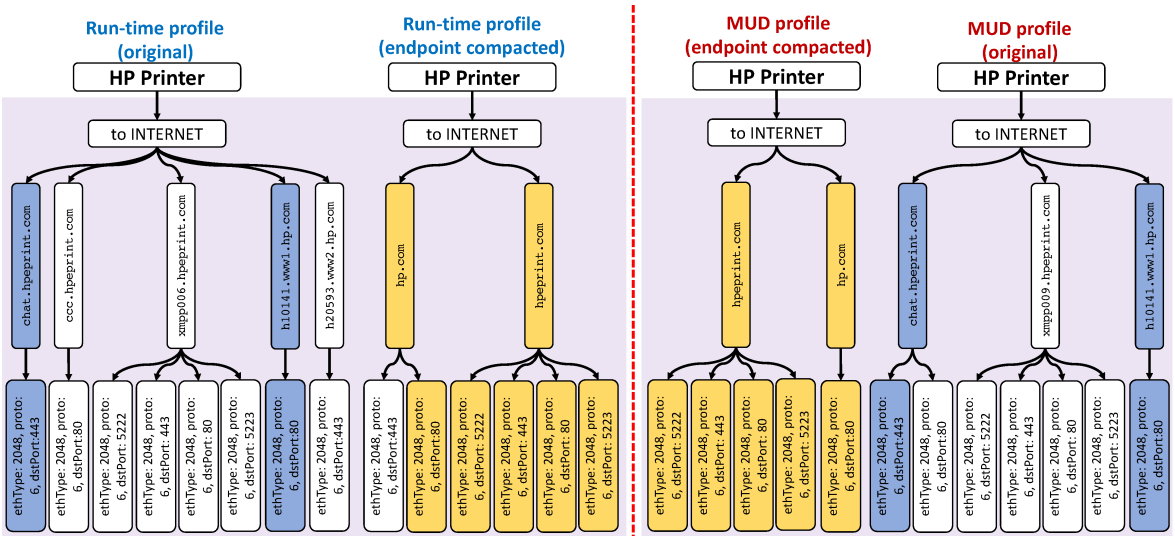


Fig. 15. Endpoint compaction of the HP printer run-time and MUD profiles in the "to Internet" channel direction yields high static and dynamic similarity (shown by the overlapping region in brown). Without compaction these similarities are significantly low (shown by the overlapping region in blue).

Fig. 14 for the iHome power plug IoT device. It can be seen that this device (in data 2016) communicates over HTTP with "`api.evrything.com`", and serves HTTP to the Local network. However, these communications do not exist in the MUD profile for the device (generated from data 2018). This difference may indicate to a network operator that a firmware upgrade is needed or that the MUD profile (offered by the manufacturer) is not complete.

Some devices (*e.g.*, the HP printer and the Hue bulb) may be found consistently in state-4 throughout the identification process. Structural variations in the profile can arise largely due to changes in the endpoints or port numbers. Tracking changes in port numbers is non-trivial. However, for endpoints fully qualified domain names can be compacted to primary domain names (*i.e.*, removing sub-domain names). If the device is under attack or compromised, it likely communicates with a completely new primary domain. Fig. 15 illustrates endpoint compaction in an HP printer profile just for the "to INTERNET" channel direction. For this channel direction and without endpoint compaction, the static and dynamic similarity scores are 0.28 and 0.25, respectively. Applying endpoint compaction results in high scores of 1 and 0.83 for static and dynamic similarities, respectively.

Endpoint compaction was applied to all of the IoT devices in the data 2016 dataset, and the results are shown under the column labelled "Endpoint compacted" in Table 4. Interestingly, this technique has significantly enhanced the identification: all state-4 devices become state-1 devices. An interesting observation here is the unknown MUD scenario for the WeMo motion detector, where the rate of incorrect identification (as WeMo switch) is fairly high, at 27.3%. However, it is not at all surprising to see different IoT devices from the same manufacturer identified as each other when compacting endpoints.

To summarize, if the identification process does not converge (or evolves very slowly), then the difference visualization and endpoint compaction described above enables network operators to discover IoT devices running old firmware.

TABLE 5
Convergence time (minutes) for all datasets & Performance metric calculated for Data-2018.

Device	Convergence Time(min)			# flows (per min)	# packets (per min)	# nodes (per min)	computing time (ms)
	Data-2018	Data-2017	Data-2016				
Amazon Echo	15	-	38355	13.72	6.58	68.83	1.38
August doorbell	60	-	45	20.11	13.44	65.84	1.71
Awair air quality	30	-	15	7.14	0.25	14.98	0.38
Belkin camera	15	1065	105	16.26	5.79	65.3	0.95
Chromecast	15	-	-	13.05	10.10	346.65	5.20
Hue bulb	15	-	9315	9.75	2.43	40.30	0.89
iHome powerplug	15	-	165	6.87	0.79	16.99	0.49
Nest smoke	15	-	15	5.30	27.00	65.70	1.55
Netatmo camera	360	-	1650	8.35	0.98	67.96	1.20
WeMo switch	15	2820	15	6.54	4.46	225.99	5.20

Attacked or compromised device: The efficacy of the process when IoT devices are under direct/reflection attacks or compromised by a botnet was also evaluated, using traffic traces collected from the testbed in November 2017 ("data 2017"), and including a number volumetric attacks spanning reflection-and-amplification (SNMP, SSDP, TCP SYN, and Smurf), flooding (TCP SYN, Fraggle, and Ping of death), ARP spoofing, and port scanning launched on four IoT devices, namely the Belkin Netcam, the WeMo motion sensor, the Samsung smart-cam and the WeMo switch (listed in Table 5). These attacks were sourced from within the local network and from the Internet. For the Internet sourced attacks, port forwarding was enabled (emulating a malware behavior) on the network gateway.

Since the IoT devices in the testbed are all invulnerable to botnets, we built a custom IoT device named "Senseme" [43] using an Arduino Yun communicating with an open-source WSO2 IoT cloud platform. This device included a temperature sensor and a lightbulb. The Senseme device was configured to periodically publish the local temperature to the server, and its lightbulb was remotely controlled via the MQTT protocol [44]. First the MUD profile of this device was generated, and then it was deliberately infected by the

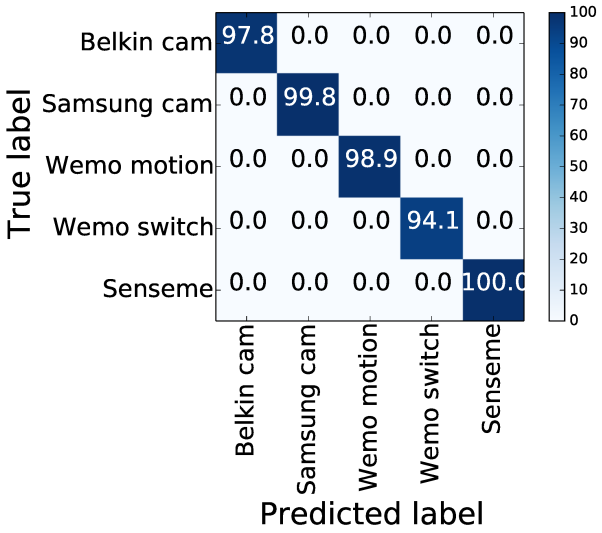


Fig. 16. Partial confusion matrix for 5 devices only (testing with attack data 2017).

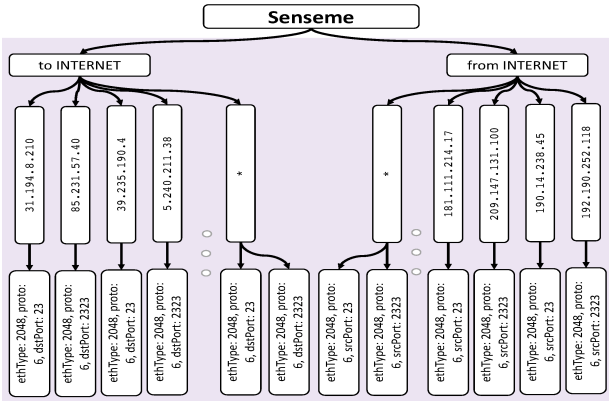


Fig. 17. Profile difference for the Mirai infected device.

Mirai botnet [45]. In order to avoid harming others on the Internet, the injection module of the Mirai code was disabled so that only its scanning module was used. A Mirai infected device scans random IP addresses on the Internet to find open ports TCP 23 and TCP 2323 for telnet access.

We applied the identification process with thresholding to data 2017, and found that all devices were identified correctly with high static similarity and low dynamic similarity (*i.e.*, high variations). A partial confusion matrix of the identification is shown in Fig. 16. Since the MUD profile of Senseme is fairly simple in terms of branch count, it quickly converges to the winner with a high static similarity score, whereas other devices require more time to converge. Therefore, the success rate for identifying Senseme device is higher than for other devices.

Various attacks have different impacts on the run-time profiles of IoT devices. For example, ARP spoof and TCP SYN would not create a new branch in the tree structure of the device profile, and consequently no variation is captured. Fraggle, ICMP, Smurf, SSDP, and SNMP attacks would result only two additional flows, meaning a minor variation is captured. However, Port scans (botnet included) cause a large variation, since an increasing number of endpoints emerge in the tree structure at run-time. For example, the Mirai botnet scans 30 IP addresses per second, causing the dynamic similarity score to approach 0. Fig. 17 shows

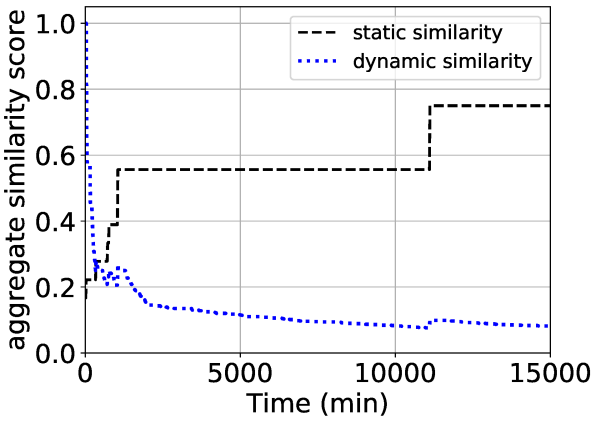


Fig. 18. Evolution of similarity scores for Belkin camera under attack.

the profile difference (or variation) for the infected Senseme device at run-time. Lastly, we show in Fig. 18 the evolution of similarity scores for Belkin camera under attack. It is seen that the static similarity slowly grows till it coverages to the correct winner – according to Fig. 16 the first row, 2.2% of instances (only during the beginning of the process) did not converge to any winner. Instead, the dynamic similarity falls in time approaching to zero.

5.3 Profile-monitoring performance analysis

We now quantify the performance of the process for real-time monitoring of IoT behavioral profiles by four metrics, namely: convergence time, memory usage, inspected packets, and number of flows.

Convergence time: Convergence time depends on user interaction with the device, the type of the device, and the similarity score thresholds. Some devices do not communicate unless the user interacts with the device (*e.g.*, the blipcare BP meter), devices like the Awair air quality sensor and the WeMo motion sensor do not require any user interaction, and devices such as cameras have many communication patterns, such as device to device, device to Internet server and remote communication. Therefore convergence times will vary based on the types of devices in the deployment.

Table 5 below lists the IoT devices and the times it took to converge to the correct device. All the devices in the 2018 dataset converged to the correct device within a day. One possible reason for this is that during the data collection, user interaction with the mobile application was programmed using a touch replay tool (*i.e.*, turning on the Hue lightbulb, checking the live camera view) in a Samsung galaxy tab, and the user interaction was replayed every 6 hours. Therefore a significant number of states of the device was captured due to these interactions, whereas with the 2017 dataset it took 2 days. The shaded cells for the 2016 data set are the devices that converged due to endpoint compaction. Other than the Netatmo camera, all other devices only converged due to compaction. For the Netatmo camera, it took 4410 minutes to converge when endpoint compaction was not applied; however due to endpoint compaction it converged within 1650 minutes.

The Smart things, Hue bulb and Amazon echo IoT devices took a considerable time to converge. When the data was analyzed, it was found that all 3 devices captured few

flows due to an interaction from the first few minutes, and then it was stale until close to the convergence time.

Three limits for the monitoring time were used, in chronological order: the first is a time limit for convergence with thresholding, then a time limit for convergence whilst compaction, and lastly a time limit to stop monitoring.

System performance: In order to quantify the performance of the system, the following four metrics were calculated: the average number of inspected packets, the average number of flows, the average number of nodes in the device profile tree, and the computation time for the compaction of the tree, redundancy removal and similarity score calculation. The average number of flows is is an important metric for the operation of a hardware switch with limited TCAM capacity, and the other 3 metrics are relevant to the scalability of the process.

As shown in Table 5, the average number of flows for each device is typically fewer than 10, with the largest flow count of about 20 for the August doorbell. This range of flow counts is easily manageable in an enterprise network setting with switches that are capable of handling millions of flow entries. However, in home networks with routers that can accommodate up to hundreds of flows, it may be necessary to limit the IoT monitoring process to only a few devices at a time, in order to manage the TCAM constraint.

Regarding the number of packets inspected, it is clear that the IoT monitoring process is very effective by keeping the number of inspected packets to a minimum (*e.g.,* mostly less than 10 packets per minute for each device). The computing time of the process solely depends on the number of nodes and the number of known MUD profiles. The time complexity of the process can be expressed as $O(n.m.log\ n)$, where n is the number of branches in the profile tree and m is the number MUD profiles we are checking against. The time complexity for the search space was reduced by employing standard hashing and binary search tree techniques. For a Chromescast device as an example in Table 5, the average computing time is 5.20 ms, where there are on average 346 nodes in its run-time profile. This can be further improved by using parallelization, whereby similarity scores are computed over individual branches. It is important to note that the computing time is upper-bounded by setting an upper bound limit on the count of tree branches generated at run-time.

Lastly, in terms of space, 40 Bytes of memory is required for each node of a tree. This means that for Chromecast, on average, less than 14 KB of memory is needed. Additionally, all known MUD profiles are present in memory. Therefore, the space complexity heavily depends on the number of MUD profiles being checked.

Limitations: Our identification approach comes with two limitations: (a) an unbounded delay in identifying devices, and (b) different types of IoT devices may have the same MUD profile (for cybersecurity applications, the knowledge of the behavioral profile is of more importance than the exact device type). We note that there exist ML (machine learning)-based models for identifying IoT devices using packet and/or flow features [29], [30]. However, ML-based methods are unable to provide exact reasons for having a low confidence in identifying devices (*e.g.,* unknown type, or behavioral changes). Another challenge with the ML-based approach is "transferability" of prediction models which often require a complete retraining to accommodate new classes of device type. In addition, a large amount of training data is required for each device type/version. Given the pros and cons of these two approaches, they are complementary.

6 Conclusion

In this paper, we have proposed a suite of tools that allow IoT manufactures to automatically generate MUD profiles while also help network operators formally check the compatibility of IoT devices with a given organizational policy prior to deployment. We have also developed a method to identify IoT devices and their behavioral changes at run-time using MUD profiles. We demonstrated using these tools and methods how the IETF MUD standard can help reduce the effort needed to dynamically identify and secure IoT devices.

References

[1] A. Hamza *et al.*, "Clear As MUD: Generating, Validating and Applying IoT Behavioral Profiles," in *Proc. ACM Workshop on IoT Security and Privacy (IoT S&P)*, Budapest, Hungary, Aug 2018.

[2] G. Sachs, "The Internet of Things: The Next Mega-Trend," [Online]. Available: www.goldmansachs.com/our-thinking/pages/internet-of-things/, 2014.

[3] J. Matherly. (2018) Shodan. https://www.shodan.io/.

[4] S. Hilton. (2016) Dyn Analysis Summary Of Friday October 21 Attack. https://bit.ly/2xCr7WN.

[5] M. Lyu *et al.*, "Quantifying the reflective ddos attack capability of household iot devices," in *Proc. ACM WiSec*, Boston, Massachusetts, Jul 2017.

[6] U. D. of Homeland Security. (2016) Strategic Principles For Securing the Internet of Things (IoT). https://bit.ly/2eXOGzV.

[7] NIST. (2016) Systems Security Engineering. https://bit.ly/2tak6fP.

[8] E. U. A. F. Network and I. Security. (2017) Communication network dependencies for ICS/SCADA Systems. https://www.enisa.europa.eu/publications/ics-scada-dependencies.

[9] FCC. (2016) Federal Communications Commission Response 12-05-2016. https://bit.ly/2gUztSv.

[10] E. Lear, R. Droms, and D. Romascanu, "Manufacturer Usage Description Specification," RFC 8520, Mar. 2019. [Online]. Available: https://rfc-editor.org/rfc/rfc8520.txt

[11] National Institute of Standards and Technology, "Securing Small-Business and Home Internet of Things (IoT) Devices," [Online]. Available: https://bit.ly/2SJMXoS, Nov 2019.

[12] European Union for Cyber Security, "Good Practices for Security of IoT," [Online]. Available: https://bit.ly/2wzuOSg, Nov 2019.

[13] Cisco DevNet. (2018) Manufacturer Usage Description. [Online]. Available: https://developer.cisco.com/site/mud/

[14] A. Hamza. (2018) MUDgee. https://github.com/ayyoob/mudgee.

[15] D. M. Mendez *et al.*, "Internet of Things: Survey on Security and Privacy," *CoRR*, vol. abs/1707.01879, 2017.

[16] F. Loi *et al.*, "Systematically evaluating security and privacy for consumer iot devices," in *Proc. ACM IoT S&P*, Dallas, Texas, USA, Nov 2017.

[17] Cisco Systems, "Cisco 2018 Annual Cybersecurity Report," Tech. Rep., 2018.

[18] S. Boddy *et al.*, "The Hunt for IoT: The Rise of Thingbots," F5 Labs, Tech. Rep., Jul 2017.

[19] V. Sivaraman *et al.*, "Smart-Phones Attacking Smart-Homes," in *Proc. ACM WiSec*, Darmstadt, Germany, Jul 2016.

[20] P. World. (2016) Backdoor accounts found in 80 Sony IP security camera models. https://bit.ly/2GbKejk.

[21] (2018) MUD maker. http://www.insecam.org/en/bycountry/US/.

[22] A. Sivanathan *et al.*, "Characterizing and classifying iot traffic in smart cities and campuses," in *Proc. IEEE INFOCOM workshop on SmartCity*, Atlanta, Georgia, USA, May 2017.

[23] A. Wool, "Trends in firewall configuration errors: Measuring the holes in Swiss cheese," *IEEE Internet Computing*, vol. 14, no. 4, pp. 58–65, 2010.
[24] D. Ranathunga *et al.*, "Case studies of scada firewall configurations and the implications for best practices," *IEEE Transactions on Network and Service Management*, vol. 13, pp. 871–884, 2016.
[25] Ranathunga *et al.*, "Verifiable policy-defined networking for security management." in *SECRYPT*, 2016, pp. 344–351.
[26] A. Basu *et al.*, *Metagraphs and their applications*. Springer Science & Business Media, 2007, vol. 15.
[27] A. Hamza *et al.*, "Combining mud policies with sdn for iot intrusion detection," in *Proc. ACM Workshop on IoT Security and Privacy (IoT S&P)*, Budapest, Hungary, Aug 2018.
[28] A. Hamza, H. Habibi Gharakheili, T. A. Benson, and V. Sivaraman, "Detecting Volumetric Attacks on loT Devices via SDN-Based Monitoring of MUD Activity," in *Proc. ACM SOSR*, San Jose, USA, Apr 2019.
[29] Y. Meidan *et al.*, "Detection of unauthorized iot devices using machine learning techniques," *arXiv preprint arXiv:1709.04647*, 2017.
[30] A. Sivanathan *et al.*, "Classifying iot devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, 2018.
[31] A. Hamza *et al.*, "Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles (Technical Report)," *ArXiv e-prints*, Apr. 2018.
[32] A. Wool, "A quantitative study of firewall configuration errors," *IEEE Computer*, vol. 37, no. 6, pp. 62–67, 2004.
[33] E. Al-Shaer *et al.*, "Conflict classification and analysis of distributed firewall policies," *IEEE JSAC*, vol. 23, no. 10, pp. 2069–2084, 2005.
[34] Cisco Systems, *Cisco ASA Series CLI Configuration Guide, 9.0*, Cisco Systems, Inc., 2013.
[35] Juniper Networks, Inc., *Getting Started Guide for the Branch SRX Series*, 1133 Innovation Way, Sunnyvale, CA 94089, USA, 2016.
[36] Palo Alto Networks, Inc., *PAN-OS Administrator's Guide, 8.0*, 4401 Great America Parkway, Santa Clara, CA 95054, USA, 2017.
[37] D. Ranathunga *et al.*, "Mgtoolkit: A python package for implementing metagraphs," *SoftwareX*, vol. 6, pp. 91–93, 2017.
[38] Ranathunga *et al.*, "Malachite: Firewall policy comparison," in *IEEE Symposium on Computers and Communication (ISCC)*, June 2016, pp. 310–317.
[39] K. Stouffer *et al.*, "Guide to Industrial Control Systems (ICS) security," *NIST Special Publication*, vol. 800, no. 82, pp. 16–16, 2008.
[40] E. Byres *et al.*, "NISCC good practice guide on firewall deployment for SCADA and process control networks," *NISCC*, 2005.
[41] D. Plonka. (2013) Flawed Routers Flood University of Wisconsin Internet Time Server. www.pages.cs.wisc.edu/~plonka/netgear-sntp/.
[42] A. Hamza. (2018) SDN pcap simulator. [Online]. Available: https://github.com/ayyoob/sdn-pcap-simulator
[43] (2018) WSO2 IoT Server. [Online]. Available: https://wso2.com/iot
[44] (2018) SenseMe. [Online]. Available: https://github.com/wso2/samples-iots/tree/master/SenseMe
[45] (2018) Mirai botnet. [Online]. Available: https://github.com/jgamblin/Mirai-Source-Code

Ayyoob Hamza received his Bachelors' degree in Computer Science from the University of Colombo, Sri Lanka and is currently a Ph.D. Candidate at the University of New South Wales in Sydney, Australia. Prior to his research career, he worked at WSO2 Inc. as a Senior Software Engineer for 3 years working on IoT solutions. His research interests includes Internet of Things, Network Security, Distributed Systems and Software-Defined Networking.

Dinesha Ranathunga is a Postdoctoral research fellow at the ARC Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS) at University of Adelaide, Australia. He received his Ph.D. for his thesis titled "Autoconfiguration of critical network infrastructure" from the University of Adelaide in 2017. His research interests include SCADA network security, Policy Defined Networking, Software Defined Networking and IoT security.

Hassan Habibi Gharakheili received his B.Sc. and M.Sc. degrees of Electrical Engineering from the Sharif University of Technology in Tehran, Iran in 2001 and 2004 respectively, and his Ph.D. in Electrical Engineering and Telecommunications from UNSW in Sydney, Australia in 2015. He is currently a lecturer at UNSW Sydney. His current research interests include programmable networks, learning-based networked systems, and data analytics in computer systems.

Theophilus A. Benson Theophilus Benson received his Ph.D. from University of Wisconsin, Madison in 2012 and his B.S. from Tufts university in 2004. He is now an Assistant Professor at Brown University in Providence, Rhode Island, USA. His research focuses on designing frameworks and algorithms for solving practical networking problems with an emphasis on speeding up the internet, improving network reliability, and simplifying network management.

Matthew Roughan obtained his PhD in Applied Mathematics from the University of Adelaide in 1994. He has since worked for the Co-operative Research Centre for Sensor Signal and Information Processing (CSSIP), in conjunction with DSTO; at the Software Engineering Research Centre at RMIT and the University of Melbourne, in conjunction with Ericsson; and at AT&T Shannon Research Labs in the United States. Most recently, he works in the School of Mathematical Sciences at the University of Adelaide, in South Australia. His research interests range from stochastic modelling to measurement and management of networks like the Internet. He is author of over a 100 refereed publications, half a dozen patents, and has managed more than a million dollars worth of projects. In addition, his coauthors and he won the 2013 Sigmetrics "Test of Time" award, and his work has featured in New Scientist and other popular press.

Vijay Sivaraman received his B. Tech. from the Indian Institute of Technology in Delhi, India, in 1994, his M.S. from North Carolina State University in 1996, and his Ph.D. from the University of California at Los Angeles in 2000. He has worked at Bell-Labs as a student Fellow, in a silicon valley start-up manufacturing optical switch-routers, and as a Senior Research Engineer at the CSIRO in Australia. He is now a Professor at the University of New South Wales in Sydney, Australia. His research interests include Software Defined Networking, network architectures, and cyber-security particularly for IoT networks.