

PROFILE-BASED SECURITY AND IN-DEPTH CHARACTERIZATION OF SMART HOME NETWORK COMMUNICATIONS

François De Keersmaecker

*Thesis submitted in partial fulfillment of the requirements for
the Degree of Doctor in Applied Sciences*

April 2026

ICTEAM
Louvain School of Engineering
Université catholique de Louvain
Louvain-la-Neuve
Belgium

Thesis Committee:

Pr. Ramin Sadre (Advisor)	UCLouvain/ICTEAM, Belgium
Pr. Cristel Pelsser (Advisor)	UCLouvain/ICTEAM, Belgium
Pr. Charles Pecher (President)	UCLouvain/ICTEAM, Belgium
Pr. Tom Barbette (Secretary)	UCLouvain/ICTEAM, Belgium
Lecturer Anna Maria Mandalari	University College London, United Kingdom
Pr. Martin Henze	RWTH Aachen University, Germany

Profile-Based Security and In-Depth Characterization of Smart Home Network Communications

by François De Keersmaecker

© François De Keersmaecker 2026

ICTEAM

Université Catholique de Louvain

Place Sainte-Barbe, 2

1348 Louvain-la-Neuve

Belgium

This work was partially supported by the F.R.S-FNRS.

*There's nowhere to go when
the sky is the limit.*

DAN SEARLE (ARCHITECTS) IN *BLACKHOLE*, 2025

Preamble

Abstract

The Internet of Things (IoT) is entering every layer of our society: factories, agriculture, health, entire cities, and homes. A domestic network which is enabled with IoT technology is dubbed a Smart Home; this paradigm is the focus of this thesis. As such devices lie close to the user and have access to private data, their security is paramount, though in practice it is mostly the opposite, as demonstrated by simple yet diabolically effective attacks targeting Smart Home devices, the most well-known being the Mirai botnet. As most of those devices exhibit straightforward and bounded communication patterns, a compelling strategy to protect them is to specify all such allowed patterns in a profile, and deny any which does not comply.

We start this thesis with a large-scale survey of datasets covering the topic of IoT network security. We analyze which protocols are popular, and which could use to be put in the spotlight. By compiling publicly accessible datasets, we hope to provide the community with a valuable asset for future research.

Afterwards, we embrace the profile-based approach to network security for Smart Home devices, and design a language to describe the aforementioned allowed communication patterns. While existing works were limited in expressiveness and flexibility, our format supports complex traffic patterns arising from interactions between multiple devices, something frequent in the context of home automation. We also implement an allow-list security system which leverages our profiles as input configuration.

Such a format raises the question of the automatic generation of profiles. Subsequently, we propose a framework which extracts the traffic patterns from the execution of Smart Home actions. We strive for a methodology more exhaustive than the state-of-the-art, and end up discovering substantially more patterns, including many which do not occur by default but only when some others are blocked. We reckon that the latter contribute to the device's robustness, by ensuring the action's success even in an inhospitable network, and consequently assess the robustness of our tested devices.

Résumé

L'Internet des objets (Internet of Things - IoT) est entré dans toutes les strates de notre société : production, agriculture, santé, villes entières, et foyers. Un réseau domestique équipé de technologies IoT est appelé une "maison connectée" (Smart Home) ; ce paradigme est le sujet de cette thèse. Puisque les appareils contenus dans ces environnements sont proches de l'utilisateur, et ont accès à des données privées, leur sécurité est primordiale, alors qu'en pratique l'opposé se fait ressentir, comme démontré par des attaques simples mais diablement efficaces ciblant ces réseaux, la plus populaire étant Mirai. Les communications émises par ces appareils étant simples et prévisibles, une stratégie pour les protéger serait de spécifier dans un profil tout motif de communication, et rejeter tous ceux qui n'y sont pas repris.

Cette thèse commence par une analyse à large échelle des jeux de données couvrant le sujet de la sécurité des réseaux IoT. Nous analysons quels protocoles sont populaires, et ceux sur lesquels l'on pourrait s'attarder un peu plus. En compilant les jeux de données accessibles publiquement, nous espérons fournir à la communauté un outil utile pour de futures recherches.

Ensuite, nous approchons la sécurité des réseaux et appareils Smart Home, et présentons un langage qui décrit leurs motifs de communication. Alors que les travaux existants proposent une expressivité et flexibilité limitées, notre format supporte des schémas de trafic complexes provenant des interactions entre différents appareils, quelque chose de régulier dans le contexte de la domotique. Nous implémentons également un système de sécurité basé sur nos profils.

Un tel format amène la question de la création automatique de profils suivant ce format. Par conséquent, nous proposons un système capable d'extraire les motifs de communication sur base d'activités d'appareils Smart Home. Notre méthode fut développée pour être plus exhaustive que l'état de l'art, et permet de découvrir substantiellement plus de motifs, incluant beaucoup qui n'apparaissent pas par défaut, mais uniquement quand d'autres sont bloqués. Ces motifs "cachés" contribuent à la robustesse d'un appareil, en assurant le succès d'une activité, même dans un réseau inhospitalier ; dès lors, nous analysons la robustesse des appareils testés.

Bibliographic notes

Journal Publications

1. F. De Keersmaecker, Y. Cao, G. K. Ndonga, and R. Sadre. “A Survey of Public IoT Datasets for Network Security Research”. In: *IEEE Communications Surveys & Tutorials* 25.3 (2023), pp. 1808–1840. DOI: 10.1109/COMST.2023.3288942.

Conference Publications

1. F. De Keersmaecker, R. Sadre, and C. Pelsser. “Supervising Smart Home Device Interactions: A Profile-Based Firewall Approach”. In: *2024 IFIP Networking Conference (IFIP Networking)*. 2024, pp. 413–422. DOI: 10.23919/IFIPNetworking62109.2024.10619760.
2. F. De Keersmaecker, R. Van Boxem, C. Pelsser, and R. Sadre. “The Forest Behind the Tree: Revealing Hidden Smart Home Communication Patterns”. In: *2025 IEEE 33rd International Conference on Network Protocols (ICNP)*. 2025, pp. 1–12. DOI: 10.1109/ICNP65844.2025.11192388.

Posters and Demos

1. F. De Keersmaecker, R. Sadre, and C. Pelsser. *Supervising Smart Home Device Interactions: A Profile-Based Firewall Approach*. Poster presented at the 8th Network Traffic Measurements and Analysis Conference (TMA 2024). May 2024.

Git repositories

- <https://github.com/smart-home-network-security/smart-home-firewall>
- <https://github.com/smart-home-network-security/pcap-fuzzer>
- <https://github.com/smart-home-network-security/smart-home-hidden-communication-patterns>
- <https://github.com/smart-home-network-security/signature-extraction>
- <https://github.com/smart-home-network-security/firewall-blocklist>
- <https://github.com/smart-home-network-security/smart-home-testbed>
- <https://github.com/smart-home-network-security/dns-unbound-cache-reader>

Remerciements / Acknowledgments

I will start this section in English for the scientific acknowledgments.

Firstly, I should thank my supervisor Ramin, for having accompanied me since even before my thesis officially started, by first proposing a research project to the FNRS, in order to fund me for 4 years. Granted, I was not the most efficient, as I have required a bit more than that to be able to wrap up the thesis. He was coming nearly every day in the office to ensure we were still alive ("How's life ?"). Around one year after I started, Cristel took part in the supervision, and even if she was only recently promoted as official supervisor, she has been playing the role since she arrived. The role of a supervisor should not be understated in research, as they both provided me with very insightful, often critical, advice that I would not have reached only by myself, whether on the core of the research or on more transversal aspects such as the writing.

I also thank all of the jury members: Charles Pecheur, Tom Barbette, Anna Maria Mandalari and Martin Henze, for the very interesting, precise yet friendly discussions during my private defense. I hope they will go in the same vein during the public one.

Finally, I thank Rémi for the good work he performed during his internship. He managed to progress on my research when I was focused on transversal stuff, and had no time to do it myself.

Il est maintenant temps de passer au français, pour les remerciements plus personnels.

Je remercie ma famille qui m'ont toujours soutenu, même s'ils ne comprennent pas trop ce que je fais. Mention spéciale pour mes parents, Poulette et Poulette, qui ont permis au Gros Poulet d'atteindre le niveau de docteur tant attendu, que ce soit de manière pratique ou plus "abstraite".

Je remercie évidemment tous les potos. Pouvoir vous côtoyer (plus ou moins) régulièrement m'a permis de garder les pieds sur terre et de penser à autre chose que la thèse. Les best bros pour littéralement tout : Lucas, Martin, Arthur, Bastien, Alex, Lilian, Matthieu. Tiphaine pour le voyage au Japon et le thé. Mérébitch et Nico pour les festivals. Les potes de Saint-Jo avec qui on a commencé nos études d'Ingé : Ysaline (aussi pour les festivals), Margaux (pour tous les projets d'info), Thibaud.

Je vis en coloc à LLN depuis 2016, j'ai donc habité avec une tonne de gens plus sympa les uns que les autres. Je leur accorde à tous une petite pensée, mais tout particulièrement à : Lola, pour m'avoir fait découvrir la teuf mon-toise ; Anatole, pour être la personne avec qui j'ai pu discuter de Pokémon le plus profondément ; Johanna, pour Mario Kart et les cinnamon rolls ; Colin, avec qui j'ai très vite vibé, et pour Smash ; Thibault, pour la pâtisserie et les tunnels sur LoL ou Dofus.

Le département INGI est un environnement de travail très particulier. Les personnes que j'y ai côtoyées ressemblent plus à une bande de potes qu'à de vrais collègues de travail, avec tous les avantages et inconvénients que cela peut apporter. En tout cas, j'y ai rencontré de merveilleuses personnes, avec qui j'ai eu l'occasion de boire beaucoup de chopes et de jouer de nombreuses parties de belotes. Je remercie donc tous les membres du département, avec certains en particulier ci-dessous. Christophe, pour les conseils muscu et les citations discutables ; Matthieu, pour les longues discussions sur la musique ; et les deux ci-dessus pour les quantités bien trop grandes de bière, et surtout les festivals, que ce soit de metal ou de techno. Dona, pour les fonds de cornets de pâtes. Ogo, pour la Taras Boulba et les pièces de théâtre. Damien, pour Toto et la gourde. Renoît Bonval, pour toutes les défaites qu'il m'a infligées à Mario Kart. Alice, pour les discussions jeux vidéo et les brunches. Mathias, pour le thé et les cafés au High Five (lol). Magali, pour les projets avant même la thèse et les discussions sur le café. Augustin, pour la Jacqueline et pour m'avoir motivé à pratiquer la course à pied. Achille, pour les jeux de cartes aléatoires à la pause de 15 :20. Aurélien, pour les tunnels (consentis) sur PoE et les 50 de blanche. Lucile, pour le voyage à Vienne. Anthony, pour avoir littéralement rendu mes expériences possibles. Alexandre, pour les bières artisanales. Michelfra, pour les soirées Mario Kart et Smash, et surtout pour tous les concerts et festivals de metal, spécialement le séjour à Londres pour aller voir Architects. Louis, pour les concerts, et puis juste pour être un mec giga sympa. Emma, pour toujours couper mes as. Tom, pour les TPs du cours de sécu. Alexander, pour les nombreux festivals. Nicolas, pour les soirées et les jeux d'ambiances (parfois discutables). Nikita, pour les tournées et la bonne humeur inconditionnelle. Maxime, pour le lait. Clément, pour le café et pour avoir écouté (et approuvé !) mon prosélytisme sur Normandie. Hélène, pour le fromage. Benoît, pour les discussions sur Martin Garrix et compagnie. Et puis évidemment, Vanessa, sans qui le département serait tombé en ruines depuis bien longtemps.

Un merci tout particulier aux personnes qui m'ont accompagné dans le a.253 pendant ces 4.5 ans. Igor ; Fabien et notre amour commun pour la progressive house, et pour m'avoir permis d'aller à Tomorrowland ; Yinan, the official Chartreuse guy ; Carole ; Patient ; Momo, qui nous fait grâce de sa présence à auteur de 2 heures par an ; Sébastien, avec qui on a bien discuté même

s'il n'est pas resté très longtemps ; Pierre, le dernier arrivé au bureau, pour sa bonne humeur contagieuse ; Rémi (encore mais maintenant c'est pour le côté perso), pour m'accompagner au quotidien depuis septembre, ce qui comprend entre autres les soirées apéro et jeux vidéo, les courses au Färm, et les Petits Biscuits au High Five ; et enfin, le meilleur pour la fin : Gorby, le GOAT (Gorby Of All Time), pour beaucoup trop de trucs, y compris les concerts de Alestorm, les fondues chinoises, et pour avoir été le meilleur binôme canard en plastique.

Ayant vécu pendant toute ma thèse à Louvain-la-Neuve, il y a quelques endroits qui sont devenus en quelque sorte les QGs. Le Brasse-Temps nous a permis de discuter de tout et de rien en buvant souvent trop de bière, voire même du jus de tomate quand j'étais dans ma période sans alcool. En tant que Belge, un de mes repas préféré est bien sûr les frites. Les friteries Jean-Lou, qui fait des très bon burgers spéciaux, et Seb, qui fait les meilleures frites et mitraillettes, m'ont permis de m'en délecter probablement trop souvent. La boulangerie MySafePlace est une super découverte (merci Achille), où on peut se poser dans une ambiance super chill (elle porte bien son nom) pour déguster un super brunch ou une variété quasi infinie de viennoiseries qui ont été des petits plaisirs de temps à autres, le cinnamon roll en tête. Finalement, impossible de ne pas parler du High Five Coffee, qui a été un témoin de première ligne de l'évolution de ma consommation de café, (quasi) inévitable lors d'une thèse. Après les simples tasses de café du bureau pour rester éveillé après le dîner, je suis vite passé à ceux de bien meilleure qualité du High Five : le classique cappuccino la majorité du temps, parfois un dirty chai ou bien un mochaccino pour le côté sucré. Par l'intermédiaire du High Five, le Père Noël m'a même encouragé pour la dernière ligne droite de cette thèse, comme illustré par l'e-mail inclus sur la page suivante. Tant que j'en suis à parler de LLN, j'accorde une mention spéciale aux oies du lac, qui sont un peu les animaux emblèmes de la ville, et que je croise très régulièrement lors de mes promenades en ville.

La musique est quelque chose de très important pour moi. J'écoute de la musique quasi tous les jours, pendant la majorité de la journée. Cette thèse a été effectuée en utilisant des heures et des heures de musique comme carburant. Aussi, il est de bon ton de remercier les artistes qui y ont contribué de manière significative. Pour le metal : Bring Me The Horizon, Parkway Drive, Architects, Normandie, Chaosbay, While She Sleeps, Bury Tomorrow, Bullet for My Valentine, Of Mice & Men, Beartooth, LANDMVRKS, et évidemment les légendes de Linkin Park (RIP Chester). Pour la musique électronique : Avicii (RIP), Eric Prydz, Swedish House Mafia, Calvin Harris, Martin Garrix, Reinier Zonneveld, HI-LO, Koven, Chase & Status, Sub Focus, Netsky, Wavedash, Pendulum et les GOATs Fox Stevenson et Madeon. Enfin, dans aucune de ces deux catégories : Imagine Dragons et les vieux de Toto.



François De Keersmaeker <francoisdekeersmaeker8@gmail.com>

Message du Père Noël - High Five specialty coffee roastery

1 message

HIGH FIVE Company <hi@highfive-coffee.be>

Tue, Dec 9, 2025 at 8:04 PM

To: "francoisdekeersmaeker8@gmail.com" <francoisdekeersmaeker8@gmail.com>

🎅 Ho Ho Ho... Bonjour mon cher François,

J'ai bien reçu ton mot, et mes lutins me l'ont lu en souriant :
tu as été *très sage cette année*, et je vois que ton souhait le plus cher...
serait un diplôme de doctorat !
Ahhh... si seulement la magie du Père Noël pouvait signer des thèses ! 🍷✨
Mais courage mon petit lutin studieux,
je suis sûr que tu es en bonne voie pour l'obtenir toi-même !

En attendant, j'ai une bonne nouvelle à t'annoncer :

🎁 Tu as été tiré au sort pour remporter un délicieux Latte à emporter !
C'est peut-être pas un doctorat...
mais ça aide très bien à tenir pendant les longues heures de recherche, ho ho ho 🍰😋

Tu peux venir chercher ton cadeau à n'importe quel moment durant tout le mois de décembre,
il te suffira de présenter ce message à l'équipe du HIGH FIVE.

Continue d'être sage (et brillant),
et bonne chance pour ton futur diplôme,
je crois en toi ! ✨

Ho Ho Ho,
Père Noël 🎅📖

--

www.highfive-coffee.be
hi@highfive-coffee.be

51 Place des Wallons
1348 Louvain-La-Neuve
Belgium

Pour décompresser le soir, je joue souvent aux jeux vidéo, principalement sur la Nintendo Switch. J'aimerais donc citer tous les jeux qui m'ont accompagné durant ce doctorat, et remercier leurs équipes créatives : évidemment, de nombreux jeux Pokémon ; Mario Kart 8 Deluxe ; Super Smash Bros. Ultimate ; Metroid Dread ; Kirby et le monde oublié ; Super Mario Bros. Wonder ; Xenoblade Chronicles ; et finalement Persona 5 Royal, qui est toujours en cours.

Je suis du genre à garder le meilleur pour la fin. La dernière personne que je veux remercier est celle qui me connaît et soutient depuis 2018, et d'autant plus durant cette thèse car il y a certains moments où j'en ai eu besoin, du soutien. Je veux bien sûr parler de Dr. Dr. Justine Dewaele. Je la remercie simplement pour être elle, et pour tout ce qu'elle fait pour moi. Because if I am her gravity, she is my oxygen.

Allez, c'est fini le blabla, it's time for science ! Bonne lecture !

Usage of Artificial Intelligence Tools

No Artificial Intelligence tool was used to generate, correct, or reformulate text. ChatGPT and GitHub Copilot were used to assist with $\text{\LaTeX}$ formatting, mainly for tables or TikZ figures. ChatGPT was used sporadically to assist in search for theoretical concepts. GitHub Copilot was used to assist with writing code in the referenced Git repositories.

List of Acronyms

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks

ACL Access Control List

ADB Android Debug Bridge

AP Access Point

API Application Programming Interface

APK Application Package

ARP Address Resolution Protocol

BFS Breadth-First Search

BLE Bluetooth Low Energy

BT Bluetooth

C&C Command & Control

CAN Controller Area Network

CHIP Connected Home over IP

CoAP Constrained Application Protocol

CPE Customer Premises Equipment

CPU Central Processing Unit

CRA Cyber Resilience Act

CSA Connectivity Standards Alliance

DCL Distributed Compliance Ledger

DDoS Distributed Denial of Service

DHCP Dynamic Host Configuration Protocol

DL Deep Learning

DM	Data Mining
DNS	Domain Name System
DODAG	Destination-Oriented Directed Acyclic Graph
DoS	Denial of Service
DPI	Deep Packet Inspection
DSL	Domain Specific Language
DTLS	Datagram Transport Layer Security
ECU	Electronic Control Unit
FN	False Negative
FODA	Feature-Oriented Domain Analysis
FP	False Positive
FSM	Finite State Machine
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input / Output
ICMP	Internet Control Message Protocol
ICNP	International Conference on Network Protocols
ICS	Industrial Control System
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IFIP	International Federation for Information Processing
IGMP	Internet Group Management Protocol

IIoT Industrial Internet of Things
IMC Internet Measurement Conference
ISM Industrial, Scientific and Medical
IoT Internet of Things
IP Internet Protocol
IPS Intrusion Prevention System
IPv4 Internet Protocol version 4
IPv6 Internet Protocol version 6
ISP Internet Service Provider
JAR Java Archive
JSON JavaScript Object Notation
LAN Local Area Network
LoC Lines of Code
LP-WAN Low-Power Wide-Area Network
LTE-M Long-Term Evolution Machine
M2M Machine-to-Machine
MAC Medium Access Control
mDNS Multicast Domain Name System
ML Machine Learning
MQTT Message Queuing Telemetry Transport
MSPL Medium-level Security Policy Language
MUD Manufacturer Usage Description
N-IDS Network-based Intrusion Detection System
NAT Network Address Translation
NB-IoT Narrow-Band Internet of Things
NIS2 2nd Network and Information Security

NLP	Natural Language Processing
NTP	Network Time Protocol
OS	Operating System
OSI	Open Systems Interconnection
PAM	Passive and Active Measurement
PAN	Personal Area Network
PCAP	Packet Capture
PFSM	Probabilistic Finite State Machine
PIR	Passive Infrared
PLC	Programmable Logic Controller
RAM	Random Access Memory
REST	Representational State Transfer
RF	Radio Frequency
RFC	Request for Comments
RNOR	Recursive Noisy-OR
RPL	Routing Protocol for Low-Power and Lossy Networks
SAAF	Static Android Analysis Framework
SCADA	Supervisory Control and Data Acquisition
SDN	Software-Defined Networking
SH	Smart Home
SNI	Server Name Indication
SQL	Structured Query Language
SSDP	Simple Service Discovery Protocol
SSIM	Structural Similarity Index Measure
TCP	Transport Control Protocol
TLS	Transport Layer Security

TMA Network Traffic Measurement and Analysis

TN True Negative

TP True Positive

UDP User Datagram Protocol

URI Uniform Resource Identifier

URL Uniform Resource Locator

VLAN Virtual Local Area Network

VPN Virtual Private Network

WAN Wide Area Network

Wi-Fi Wireless Fidelity

WLAN Wireless Local Area Network

WSN Wireless Sensor Network

XML Extensible Markup Language

YAML YAML Ain't Markup Language

Contents

Preamble	i
Remerciements	iv
Usage of Artificial Intelligence Tools	ix
List of Acronyms	xi
Table of Contents	xvii
1 Introduction	1
Structure	5
2 Generic Concepts of IoT, Smart Home & Network Security	9
2.1 Internet of Things	10
2.1.1 Network Topologies	10
2.1.1.1 Tree	11
2.1.1.2 (Full-)Mesh	11
2.1.1.3 Star	11
2.1.2 Generic communication protocols	12
2.1.2.1 Network layer	13
2.1.2.2 Transport layer	14
2.1.2.3 Application layer	14
2.1.3 IoT-specific protocols	17
2.1.3.1 Physical and Data Link layers	17
2.1.3.2 Network layer	18
2.1.3.3 Application layer	19
2.1.3.4 Others	21
2.1.4 IoT subdomains	21
2.2 Network security	22
2.2.1 Vulnerabilities	23
2.2.2 Security systems	24
2.2.2.1 Binary classifiers	24
2.2.2.2 Position	25
2.2.2.3 Decision logic	26
2.2.2.4 Usage of state	26

- 2.2.2.5 Concrete security systems 27
 - 2.2.2.6 Choice of security features 28
 - 2.3 Smart Home 28
 - 2.3.1 Smart Home network topology 29
 - 2.3.2 Security in Smart Home networks 30
- 3 Existing Public Datasets for Research in IoT Network Security 35**
 - 3.1 Introduction 36
 - 3.1.1 Scope 37
 - 3.1.2 Contribution 39
 - 3.1.3 Structure 39
 - 3.2 Related Work 39
 - 3.3 Methodology 42
 - 3.4 Dataset Characteristics 44
 - 3.4.1 Traffic type, data content, presence of attacks 44
 - 3.4.2 Scale 47
 - 3.4.3 Collection methodology 47
 - 3.4.4 Year 48
 - 3.4.5 Presence of metadata 48
 - 3.4.6 Availability and usage by others 48
 - 3.4.7 Related research perspective 49
 - 3.5 Overview of datasets 50
 - 3.5.1 Summary of research perspectives 50
 - 3.5.1.1 Protocol distribution 50
 - 3.5.1.2 Number & category of devices 52
 - 3.6 Challenges and Future Directions 52
 - 3.6.1 IoT-specific protocols and attack traffic 52
 - 3.6.2 Data privacy and confidentiality 54
 - 3.6.3 Scale and complexity 55
 - 3.6.4 Realistic traffic generation 56
 - 3.7 Conclusion 57
- 4 Construction of a Realistic Smart Home Testbed 59**
 - 4.1 Devices 59
 - 4.1.1 End devices 60
 - 4.1.1.1 Power plugs 60
 - 4.1.1.2 Light bulbs 60
 - 4.1.1.3 Cameras 61
 - 4.1.1.4 Other sensors 61
 - 4.1.2 Hubs 61
 - 4.2 Smartphone apps 62
 - 4.3 Complete network 63

4.4	Interactions	65
4.4.1	Interactions per device	65
4.4.2	Automation of interactions	66
4.4.2.1	Tools: Android Debug Bridge & Maestro	66
4.4.2.2	Verification of the interaction's outcome	66
4.5	Conclusion	67
5	A Profile-Based, Cross-Device Interaction-Enabled Smart Home Firewall	69
5.1	Introduction	69
5.2	Motivation and threat model	71
5.2.1	Smart Homes and device interactions	71
5.2.2	Device profiling: the MUD format	72
5.2.3	Limitations of MUD and potential attacks	74
5.2.3.1	Sequential dependency between patterns	74
5.2.3.2	Traffic statistics	75
5.2.3.3	Protocols	76
5.2.4	Threat model	77
5.3	System overview and description	77
5.3.1	System overview	77
5.3.2	Devices profiles	79
5.3.3	The Firewall	82
5.3.4	Finite State Machine for interactions	83
5.3.5	The Translator	84
5.4	Experimental setup	84
5.4.1	Devices	84
5.4.2	Traffic capture	86
5.4.3	Profile creation	86
5.5	Evaluation	87
5.5.1	Fuzzing-based validation	87
5.5.2	Validation on attack-free scenarios	89
5.5.3	Validation on attack scenarios	91
5.5.4	CPU and Memory usage under stress	94
5.6	Discussion and related work	95
5.6.1	Lightness of the security system	95
5.6.2	What types of attacks can/cannot be blocked?	96
5.6.3	Level of abstraction of profiles	97
5.6.4	Creation and distribution of profiles	99
5.6.5	Extensions to the MUD standard	101
5.7	Conclusion	101

6 In-depth Profiling and Robustness Assessment of Smart Home

Events	103
6.1 Introduction	103
6.2 Problem statement and scope	107
6.3 Description of the framework	109
6.3.1 General workflow	109
6.3.2 Interaction trigger	111
6.3.3 Event traffic capture	111
6.3.4 Event success verification	112
6.3.5 Event signature extraction	112
6.3.5.1 Replacing IP addresses by domain names . .	112
6.3.5.2 Aggregating packets to Flow IDs	113
6.3.5.3 Extracting the relevant Flow IDs	114
6.3.6 Event signature tree definition	114
6.3.7 Flow ID firewalling	115
6.3.8 Tree pruning heuristics	115
6.3.8.1 Path pruning	116
6.3.8.2 Node pruning	116
6.4 Testbed and Example	117
6.4.1 Experimental setup	117
6.4.2 Example signature trees	120
6.5 Empirical evaluation of the framework and pruning heuristics	121
6.5.1 Exhaustiveness of the pruning heuristics	121
6.5.1.1 Total number of unique Flow IDs	122
6.5.1.2 Number of Flow IDs extracted per tree depth	123
6.5.2 Protocol usage	123
6.5.2.1 <i>Boot</i> events	124
6.5.2.2 Other, non- <i>boot</i> events	124
6.5.3 DNS data: domain names & DNS resolvers	126
6.5.3.1 <i>Boot</i> events	126
6.5.3.2 Other, non- <i>boot</i> events	127
6.5.3.3 Discussion	128
6.6 Assessment of events' robustness	128
6.6.1 Number of <i>hidden</i> Flow IDs	129
6.6.1.1 <i>Boot</i> events	130
6.6.1.2 Other, non- <i>boot</i> events	131
6.6.1.3 Grouped per category	131
6.6.1.4 Discussion	133
6.6.2 Number of Flow IDs blocked until failure	134
6.6.3 Failure causal attribution to Flow IDs	136
6.6.4 Failure probability per protocol	136
6.6.5 Failure probability when blocking a set of Flow IDs . .	138

6.6.5.1	Derivation of the formula	138
6.6.5.2	Computation	139
6.6.6	Takeaway: robustness guidelines	140
6.6.6.1	Various <i>hidden</i> Flow IDs	140
6.6.6.2	Various DNS resolvers	142
6.6.6.3	Using domain names instead of direct IP addresses	142
6.7	Comparison of results	143
6.7.1	Number of Flow IDs extracted	143
6.7.1.1	Interaction-level profiling	143
6.7.1.2	Device-level profiling	144
6.7.2	Domain names extracted	145
6.8	Related work	146
6.8.1	Smart Home traffic signature extraction	146
6.8.2	Empirical traffic measurements	149
6.8.3	Chaos Engineering	150
6.9	Conclusion	151
7	Discussion & Perspectives	153
7.1	Automatic profile generation	153
7.1.1	Note about data purity	160
7.2	Model of the event failure probability	160
7.3	Geolocation-based traffic divergences	161
7.3.1	Contacting the closest servers	161
7.3.2	Regional privacy rules	163
7.3.3	Implementation insights	164
7.4	Clear-box approach to profiling	165
7.4.1	Device firmware	165
7.4.2	Companion app APK	165
7.4.3	Takeaways	167
8	Conclusion	169
A	Appendix	207
A.1	Summary of all surveyed datasets	207
A.2	Complete device profiles	217
A.2.1	Manufacturer Usage Description (MUD)	217
A.2.2	Our format	220
A.3	Firewall-incurred latency: split per device	221

List of Figures

2.1	Architecture of a typical IoT deployment	10
2.2	End devices sub-network topologies	11
2.3	OSI layered model	12
2.4	Sample confusion matrix	25
2.5	Example Smart Home network	30
2.6	Example Smart Home network with two VLANs	31
2.7	Example Smart Home network with host-based security	32
2.8	Example Smart Home network with network-based security	33
3.1	Methodology for datasets gathering	42
3.2	Number of published IoT-related datasets per year	43
3.3	Feature-Oriented Domain Analysis: legend	44
3.4	Feature diagram of dataset characteristics: first level	45
3.5	Feature diagram of dataset characteristics: Traffic type	45
3.6	Feature diagram of dataset characteristics: Data content	45
3.7	Feature diagram of dataset characteristics: IoT subdomain	46
3.8	Feature diagram of dataset characteristics: Collection method- ology	46
3.9	Feature diagram of dataset characteristics: Research perspective	46
3.10	Distribution of protocols in IoT-related datasets	51
3.11	Number & category of devices used per dataset	53
4.1	Complete experimental Smart Home network	64
5.1	Example Smart Home network	72
5.2	Example Smart Home network showcasing device interactions	72
5.3	Example Smart Home network, including attacks	74
5.4	Overview of the system	78
5.5	Minimal device profile example	79
5.6	General NFTables/NFQueue firewall operation	82
5.7	Interaction FSM	84
5.8	Subset experimental Smart Home network	85
5.9	Profile size: Number of policies and interactions, per device	87
5.10	Firewall size: Number of NFTables rules and NFQueues, per device	88

5.11	Lines of Code of NFQueue source code, per device	88
5.12	Latency incurred by the AP hosting the firewall, in attack-free scenarios	90
5.13	Firewall latency per category of processing intensity	92
5.14	Latency induced by the AP hosting the firewall, per attack scenario	93
6.1	Example SH network involving a smart lamp	108
6.2	Overview of the profiling workflow	110
6.3	Firewall configuration based on Flow IDs contained in the tree	115
6.4	Tree pruning heuristics (considering BFS traversal)	116
6.5	Subset experimental Smart Home network	118
6.6	Event signature tree for the Tapo P110's <i>toggle</i> event	120
6.7	Total number of unique Flow IDs discovered, for both pruning heuristics	122
6.8	Number of Flow IDs discovered per tree depth, for both pruning heuristics	123
6.9	Number of unique domain names and DNS resolvers contacted, for each device's <i>boot</i> event	127
6.10	Number of <i>hidden</i> domain names contacted, for both pruning heuristics, for non- <i>boot</i> events	128
6.11	Number of unique Flow IDs, for each <i>boot</i> event	130
6.12	Hidden Flow ID ratio for both pruning heuristics	131
6.13	Hidden Flow ID ratio, for <i>path pruning</i> , grouped per device category and companion app	132
6.14	Path lengths until event failure	135
6.15	Event failure probability per blocked protocol	137
6.16	Probability of event failure when blocking a random set of Flow IDs	141
6.17	Comparison of discovered unique Flow IDs between two related works and our two pruning heuristics, at the interaction level.	144
6.18	Comparison of discovered unique Flow IDs between MUDgee and our two pruning heuristics, at the device level.	145
6.19	Venn diagrams showing the domain names extracted by Mandalari <i>et al.</i> and our study	147
7.1	Comparison between basic Noisy-OR and Recursive Noisy-OR models	162
A.1	Latency incurred by the AP hosting the firewall, in attack-free scenarios, split per device	222

List of Tables

3.1	Related surveys (sorted by date)	40
3.2	Datasets classified per research perspective	51
4.1	Companion apps used for each device	64
4.2	Verification strategy per device	67
5.1	List of protocol fields, per layer, which can be leveraged by the firewall for its verdict	81
5.2	Testbed devices	85
5.3	Mean of performance metrics	95
6.1	Devices in our testbed, with corresponding apps and interactions	119
6.2	Description of the Flow IDs depicted on Figures 6.6a and 6.6b	120
6.3	Number of total & <i>hidden</i> Flow IDs per protocol, for <i>boot</i> events only	124
6.4	Number of total & <i>hidden</i> Flow IDs per protocol and per pruning heuristic, for non- <i>boot</i> events	125
6.5	Related works' device and interaction coverage.	143
6.6	Number of domain names extracted by Mandalari <i>et al.</i> and our study	146
7.1	Abstraction levels of Smart Home profiling	155
A.1	IoT-related datasets involving IP-based protocols	209
A.2	IoT-related datasets involving IoT-specific protocols	212
A.3	Non-IoT datasets	215

Introduction

| 1

In 2016, the world was struck by the *Mirai* malware [5], as it performed the largest Distributed Denial of Service (DDoS) attack to date, peaking at one Tbps. To obtain such a large traffic rate, the malware operated by reproducing and propagating itself over the Internet to vulnerable devices. While general-purpose computers are usually protected against this kind of threat, those were not the target of the malware. Rather, it aimed for Linux-based IoT and embedded devices which were left with default credentials. This procedure, while seemingly simplistic, was tremendously effective to infect such devices, as demonstrated by the high figures of the attack. It managed to shut down well-known websites and services, including the cybersecurity blog Krebs on Security [6], french cloud provider OVH [7], and former Domain Name System (DNS) provider Dyn (now acquired by Oracle [8]).

This example is the first occurrence of a worldwide attack specifically involving IoT devices. In the context of this attack, such devices posed as the victims, as they were infected by the botnet malware, but also as the attackers, as the army of infected devices was used as a source of crippling heavy traffic. Nonetheless, it is far from being the last. As late as 2024, *Mirai* itself reoffended with a DDoS attack peaking at 5.6 Tbps [9]. Besides, taking inspiration on *Mirai*, new variants are developed regularly, succeeding attacks always larger than the previous one. The newest one is the *Aisuru* botnet, which managed, in 2025, a DDoS attack peaking at 11 Tbps [10], a rate more than ten times larger than the initial *Mirai* attacks, then another one exhibiting a rate nearly as tremendously high as 30 Tbps [11], by far the largest DDoS attack yet. Such attacks are real-world examples which show that the security of IoT devices is still neglected, nearly ten years after the first occurrences.

Nowadays, the IoT has entered virtually all facets of society. People use it in cities, in factories, in the health sector, in agriculture, and in homes, the latter being the focus of this thesis. The gap between the IoT ubiquity and its lack of security has become inadmissible, and has motivated official bodies to take action. The recent European directives 2nd Network and Information Security (NIS2) [12] and Cyber Resilience Act (CRA) [13] now urge companies, among other requirements, to ensure sufficient security in their computer systems, including IoT devices. One of the measures of the latter, related to the aforementioned botnet attacks, is the obligation to implement

security by default: easy-to-guess default passwords are now prohibited. In the near future, legal action will be enforced over companies which do not comply with the requirements. More than ever, the time is now favorable to advancements regarding any aspect of IoT security.

Most devices that got infected with the aforementioned malware are Internet-connected cameras, devices which are installed with the incentive of, ironically, protecting a household. The malware managed to do so because the users, which are usually not tech-savvy, left the devices configured with the default credentials. Such cameras fall into the category of the SH, a popular subdomain of the IoT. Its purpose is to apply the concepts of the IoT to a household to automate everyday tasks, aiming to provide more convenience to the inhabitants. To do so, everyday household objects are augmented with communication capabilities, including said cameras, as well as small objects such as power plugs or light bulbs, to big appliances such as fridges; a wide range of objects which potentially suffer from the same vulnerabilities as the exploited cameras. The popularity of the SH paradigm has risen tremendously in the last years: In the USA alone, the number of households using SH devices was estimated at nearly 70 million in 2024, which accounts for nearly 50% of all households in the country [14]. The fact that the SH represents such a wide potential target is urging for improvements in the security of such environments.

The attacks used as introduction, while having a dramatic effect of the global Internet traffic and being able to shut down even large services, are quite distant for the regular SH user. However, they might also have very concrete effects on the user's vicinity. First, a security vulnerability might translate to a safety issue: if a smart lock is compromised, an attacker could unlock it to break into the house. Then, as SH devices are present in the user's personal space, they are more likely to have access to potentially private data. In that case, the privacy of such data is tied to the security of the devices handling them. An insecure device means that unauthorized parties might access private data [15, 16]. Actually, there does not even need to be a hacker involved, as it has been shown that SH providers themselves have eavesdropped on their customers' private data [17, 18, 19]. To ensure it stays private, the security of the devices themselves and of their communications should be strengthened, and unwanted communication should be flagged.

Numerous ways exist to implement security inside a network, often striking a compromise between proficiency and resource consumption. One of the most basic is the firewall: a system which, based on a set of rules describing the network packets or flows which should be accepted or rejected, applies said verdict. General-purpose devices, among others computers and smartphones, as their name suggests, are designed to perform any task one could expect from a computing device, *e.g.* browsing the internet, or running

software of any kind (working, gaming, etc.). As such, regarding network communication, their behavior is unbounded, and they are expected to issue virtually limitless communication patterns. For those, a firewall should be configured as a *deny-list*: accepting by default any traffic, and rejecting only explicitly described traffic, *e.g.* to deny access to unauthorized hosts, or limiting the traffic rate. On the contrary, IoT devices are often designed for a specific task, and generally do not need to perform anything else. As such, their communication patterns are way more restricted [20], so much so that the inverse security strategy can be used, the *allow-list* strategy. With the latter, the firewall is provided with a list describing the traffic it should accept, and any other traffic should be rejected. This would obviously be intractable for general-purpose devices, but is quite effective for devices expected to only issue a finite set of communication patterns.

The contributions presented in this thesis will focus on this kind of purpose-driven devices, which exhibit a finite, easily identifiable set of functionalities, *e.g.* power plugs and light bulbs. With the evolution of technology, SHs now also include more complex devices running a full-fledged Operating System (OS), and providing a large, potentially limitless feature set, extensible thanks to third-party app stores, such as smart TVs and speaker hubs (*e.g.* the Alexa-powered Amazon hubs). As their features are limitless, so are the communication patterns they issue, which renders an analysis with a bounded profile impossible. Therefore, those devices are considered out of scope.

Early work in the definition of a format describing the network patterns of IoT devices was carried out by Barrera *et al.* [21], who proposed a standard format to express the traffic allowed for a given device, in a file that we will call the device's *profile* in the remaining of this manuscript. As a proof-of-concept, they implemented an allow-list firewall leveraging their handcrafted profiles. The concept for such a format would afterwards be retrieved by the Internet Engineering Task Force (IETF), an organization which standardizes protocols globally used on the Internet. Backed up by leading companies such as Google and Cisco, they standardized, as their Request for Comments (RFC) document numbered 8520, the Manufacturer Usage Description (MUD) format, with a goal equivalent to the aforementioned research work.

The formal definition of this standard incentivized researchers to design systems able to automatically generate traffic-describing profiles, whether strictly complying with MUD [22] or simply taking inspiration from it [23, 24, 25, 26]. However, even if this standard is a big step towards a better security for IoT networks, it remains sparsely used in practice. One of the reasons might be that, due to the fast-paced nature of technological advancements, the format is already quite obsolete, and cannot precisely express the complexity of today's IoT networks. The main hurdle is that MUD and similar formats have been designed as if IoT devices act individually, and issue singular, in-

dependent communication patterns. However, this is untrue in substantial applications of the IoT, for instance in a SH. In the latter, devices are often required to interact with each other and cooperate [27], forming what is known as a *routine*. Here is a simple example: the light in the corridor should turn on whenever the motion sensor detects motion. In practice, this means that a pattern issued by device *B*, here the light, depends on a previous pattern issued by device *A*, the motion sensor. Even considering a single device, some traffic patterns are inherently in the form of a sequence of network flows, each depending on the previous one: for example, a device can only contact a remote server if it previously resolved its domain name via a DNS query. In this thesis, we will design a format which builds on the concepts of MUD, while overcoming its stated shortcomings. More formally, we formulate our first research question:

Research Question 1

How to design a format to express dependencies between Smart Home communication patterns ?

The main purpose of a profile of a device's expected behavior is as ground-truth for a security system; basically, the latter will accept what is specified in the profile, and reject what is not. In such cases, the profile should be exhaustive with respect to the device's actual behavior: it should specify all legitimate traffic, to ensure it will be accepted, even if some traffic patterns occur less frequently than others. If legitimate traffic is not referenced in the profile, it will be wrongly rejected: this is known as a *false positive*. Existing works which have tackled the task of automatically producing device profiles all share a caveat in their design: they only profile the superficial communications of the device, and miss hidden patterns which only occur when actively tampering with the traffic. This results in shallow profiles which systematically miss a noticeable part of the device's expected traffic, which would be rejected if it appeared during the SH system's operation. This research gap has us raise the following research question:

Research Question 2

How to exhaustively extract communication patterns issued by Smart Home devices ?

Such *hidden* communication patterns are interesting because they do not appear during typical operation, but only whenever the default patterns are blocked. They allow communication to carry on even if its default form is prevented to do so. Therefore, we reckon that hidden patterns embody a sort of

robustness of the communication. Recall that the purpose of the SH paradigm is to provide convenience to the user by automating household tasks, or routines. Such routines require several devices to cooperate, and if only one fails, it disrupts the whole routine. The robustness of SH devices is therefore primordial. This leads us to yet another research question:

Research Question 3

To what extent do hidden communication patterns embody the robustness of Smart Home communication ?

This thesis covers various aspects of the network communication of SH devices, from the automated creation of a profile encompassing all the device's intended communication patterns, to the application of this profile as configuration for a network security tool, providing the network with tailored security, and even the assessment of the robustness of said communication. While they were designed focusing on this specific environment, the scope of the concepts can be widened to be applied to any device which exhibits a limited set of predictable network communication patterns, *e.g.* in a Wireless Sensor Network (WSN) or an Industrial Control System (ICS).

Structure

Before starting the main body of this thesis, this section will summarize the manuscript's structure.

Chapter 2: Background The first chapter after this introduction describes all background concepts required to understand the content of the manuscript and the scientific contributions. The big concepts explained are the Internet of Things as a whole, notably including a presentation of relevant communication protocols, followed by a description of basic solutions for network security, and finally a deeper dive in the IoT subdomain which is the Smart Home. This chapter does not present state-of-the-art and related scientific works; they will be referenced further, in the relevant chapters.

Chapter 3: Survey of public datasets Any good research requires data serving as the base for a sound evaluation. Before tackling the aforementioned research questions, we performed an in-depth survey of existing public datasets related to network security, mainly focused on IoT-related datasets, with the objective of identifying ones that could be relevant for our own evaluations. The outcome of that chapter was a publication in the journal *IEEE Surveys & Tutorials* [1]. However, due to the active and complex nature of our

contributions, no found dataset was fit for the job, thus we ended up building our own SH testbed environment, in order to record our own traffic.

Chapter 4: Smart Home testbed For the evaluation of our original contributions, we necessitated a sample network which mimics as closely as possible an actual SH environment. That chapter details the construction of such a testbed network.

Chapter 5: Smart Home Devices Profiling & Firewall Chapter 5 is the first one detailing an original contribution. Its objective is to answer Research Question 1. We propose a novel Domain Specific Language (DSL) with the same objective as MUD and similar incentives, *i.e.* expressing the expected communication patterns of SH devices, but enhanced with respect to existing work. While existing work only consider patterns tied to a single device, we designed our format such that it can express patterns involving multiple devices cooperating to perform a single action. Additionally, we added support for neglected protocols and traffic statistics, to cover a wider range of behaviors. Our language can then be used to produce a profile for a SH device, which can in turn serve as configuration for a security system. As a proof-of-concept, we incorporate this language in an end-to-end framework which, based on a profile, produces the necessary configuration files to deploy an allow-list firewall tailored to the device's network behavior. This contribution was presented at the 2024 IFIP Networking conference [2].

Chapter 6: Smart Home Events In-Depth Profiling & Robustness In Chapter 6, we address the problem of obtaining an exhaustive profile of a SH device's communication patterns, and tackle Research Question 2. As a solution, we created a framework able to extract the patterns issued when a device performs a given action, *e.g.* turning a smart plug on or off. Our framework was designed to be more exhaustive than existing approaches, by also considering hidden patterns. Our methodology is to iteratively block the observed patterns, with the intent of making new, hidden ones appear. To do so, we leverage and slightly modify the firewall system described in Chapter 5. The extracted profiles comprise a broader and more exhaustive set of network patterns, including hidden ones which have been systematically missed by previous work. As those embody the robustness of the device's communication, we address Research Question 3 by performing a data-driven assessment of the robustness of SH devices, based on empirically defined metrics. This work was first presented at the 2025 International Conference on Network Protocols (ICNP) [3]. Afterwards, we decided to design an even more in-depth exploration of the devices' traffic, and to push further the assessment

of the framework and evaluation of the observed SH communications. An extended version is currently in preparation for a journal submission.

Chapter 7: Discussion & Perspectives This thesis addresses existing issues in the scientific community. However, no solution is perfect. In Chapter 7, we take a step back, and discuss the shortcomings of our solutions, while pinpointing potential routes for improvement. Notably, we investigate if a clear-box approach to the device profiling could be applicable in our scenario, as opposed to the opaque-box approach we had been following.

Chapter 8: Conclusion Chapter 8 concludes the thesis by summarizing our contributions.

Generic Concepts of IoT, Smart Home & Network Security

2

The computing paradigm dubbed the Internet of Things (IoT) has undergone a meteoric rise in the last decade. Ironically, this might be one of the most loosely defined concepts in computing. In this chapter, we will attempt giving a definition and description of the IoT, at least in the context of this thesis. We will also delve deeper on one of the most popular subdomains of IoT, namely the Smart Home (SH), as it is the part we focused on in this thesis.

Those concepts showcase how our world strives towards a complete digitalization. While creating new opportunities, this also opens new issues. As illustrated by the NIS2 European directive [12], a primordial one is the security of such digital systems. This concern is also central to this thesis; therefore, we will present basic concepts tied to IoT and SH network security.

This chapter is structured as follows:

- Section 2.1 defines the IoT as a general concept, and presents its common characteristics:
 - Typical network topologies used in the context of IoT are described in Section 2.1.1.
 - Section 2.1.2 lists generic TCP/IP protocols commonly used in IoT networks, while Section 2.1.3 introduces protocols specifically designed for IoT communication.
 - Section 2.1.4 skims over several subdomains of IoT.
- Section 2.2 presents the current unsatisfactory state of network security in IoT networks:
 - Some reasons for this limited security are presented in Section 2.2.1.
 - Existing systems to provide network security in the context of IoT are discussed in Section 2.2.2.
- The Smart Home, the subdomain of the IoT which is the focus of this thesis, is presented in more detail in Section 2.3. We specify the aforementioned concepts for the SH context:

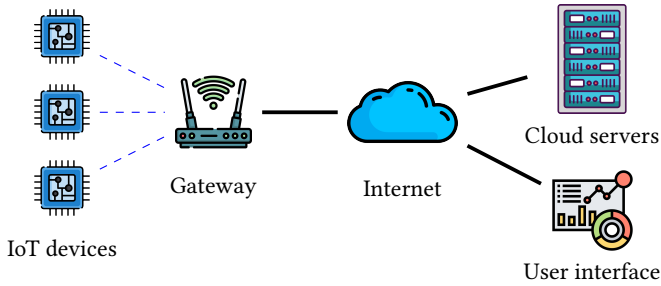


Figure 2.1: Architecture of a typical IoT deployment

- The network topology generally used for SH networks is described in Section 2.3.1.
- The strategies to secure a SH network are presented in Section 2.3.2.

2.1 Internet of Things

The term “Internet of Things” stands for a set of technologies that allow to equip physical objects with sensors and processing ability and to let them exchange information with other devices over communication networks [28]. Figure 2.1 shows the architecture of a typical IoT deployment consisting of the following components:

1. *IoT devices* sense, and sometimes also act, in the physical world. They send data to and receive commands from servers and other devices.
2. In order to reduce energy consumption and save resources, IoT devices often use tailored communication technologies and protocols [29] (see Section 2.1.3). If those are not compatible with IP-based networks, a *gateway* acts as a translator between the IoT network and the Internet.
3. *(Cloud) servers* located in the Internet receive and analyze the data sent by the IoT devices.
4. *Human-machine user interfaces*, such as web-based dashboards or mobile companion applications, allow the users to inspect the data and control the devices.

2.1.1 Network Topologies

Depending on its purpose, an IoT network can be organized in varying topologies [30]. Three of them will be described in the following, namely the tree, mesh, and star topologies. All three are depicted on Figure 2.2.

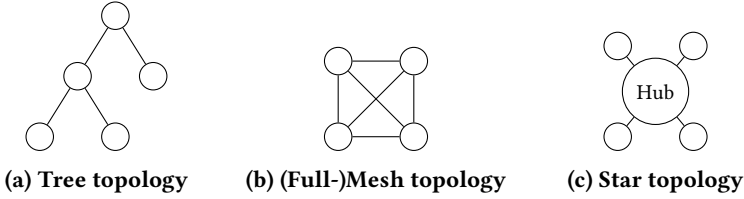


Figure 2.2: End devices sub-network topologies

2.1.1.1 Tree

The tree-shaped topology, shown on Figure 2.2a, is analogous to tree data structures, common in computing science. It is used when the network adopts a hierarchical structure. The *root* of the tree is the parent of all other nodes. Then, each node has a parent, and potential children. This topology is commonly used for multi-hop networks which necessitate message translation for protocol interoperability, such as Zigbee networks [31] (described in Section 2.1.3.2). In such cases, the network's root takes the role of the translating gateway. When a generic node issues a message, it forwards it to its parent, which in turn forwards it to its own parent, and so on until it reaches the root.

2.1.1.2 (Full-)Mesh

A mesh network, shown on Figure 2.2b, contrarily to a tree-shaped network, provides a flat structure, without any hierarchy. Connections are created between nodes that must communicate directly with each other. A mesh network where every node is connected with every other, is called a full-mesh network. WSNs deployed over a reduced area might apply this topology [32].

2.1.1.3 Star

A star-shaped topology, illustrated on Figure 2.2c, can be considered as a corner case of a tree-shaped network, with a depth of only one, and numerous children connected to the root. In such a network, all end devices are connected to a central node, which acts as a hub: it receives traffic from all nodes. As such, it is the *de facto* standard topology for SH networks [33, 34, 35], which often require a form of centralized control. The hub node can simply be the network's Internet gateway, or a device specifically designed to provide such functionality. As for a tree-shaped network, the hub, being the network's root node, can provide message translation between two different protocols, *e.g.* between an IoT-specific protocol such as Zigbee, and the IP-based Internet.

Number	Layer	Protocol(s)
7	Application	HTTP, DNS
6	Presentation	TLS
5	Session	SOCKS
4	Transport	TCP, UDP
3	Network	IP
2	Data Link	Ethernet, Wi-Fi
1	Physical	802.11

Figure 2.3: OSI layered model

2.1.2 Generic communication protocols

One of the quintessential features of IoT devices is their capability to communicate: with each other or the Internet; to exchange data or be remotely controlled, etc. [28]. Generally, digital communication between computing devices broadly follows the Open Systems Interconnection (OSI) layered model [36], depicted on Figure 2.3, which is a theoretical framework designed to facilitate interconnection and communication between systems. It is composed of seven layers, numbered from one to seven; the higher the number, the more abstract the layer is. Each layer leverages the features offered by the layer below it, and adds its own features, to provide the layer's intended functionality [37]. For instance, the TCP protocol [38], located at the *Transport* layer (layer 4), uses the IP layer [39], located at the *Network* layer (layer 3), for routing over the Internet. The former then guarantees communication reliability, by providing packet retransmissions and reassembling, among others. It should be noted that this model is only theoretical, and that practical networks might depart from the strict seven-layers structure. In practice, the lines might be blurred between two consecutive layers, and a single protocol might implement features from several layers, or two stacked protocols might be considered part of the same layer [40].

To perform network communication, general-purpose devices (PCs or smartphones) use the typical protocols from the TCP/IP stack [41]. Most of those were standardized by the IETF, as part of their series of RFC documents. IoT devices which provide the hardware support for those protocols tend to use them as well. This section will quickly go over the IP-based protocols and their relevant fields which will be considered by our upcoming contributions, ordered by increasing layer level.

2.1.2.1 Network layer

Internet Protocol (IP) IP is used to address hosts over the Internet, all around the planet. Contrarily to Medium Access Control (MAC) addresses, IP addresses are able to address a host located in a different subnet. The IP header contains the *source* and *destination* IP addresses of the packet. The packet is routed over the Internet based on the destination address. By itself, IP does not provide reliable delivery of packets, and packet losses can occur¹. Reliable delivery is handled by the upper layer, usually thanks to TCP.

The most widespread version of IP is Internet Protocol version 4 (IPv4) [39]. Although Internet Protocol version 6 (IPv6) [43] should supersede it nowadays, it still lacks global traction, even more so in IoT networks [44].

Internet Control Message Protocol (ICMP) ICMP [45] is a control-plane protocol, used to transmit messages relative to the operational status of the network. The main field, and the only which we will consider, is the *type*, which encodes the operation the message is supposed to perform. A popular use is to test the reachability of another host. To this end, an ICMP *Echo Request* is sent. If the destination is reachable, it might respond with an *Echo Reply*. However, such messages are often prevented from progressing in the network; in this case, a *Destination Unreachable* message is sent as a response.

Internet Group Management Protocol (IGMP) The basic transmission strategy for a packet is using *unicast*: the packet has one, and only one, destination. However, some use cases require sending the same packet to several destinations. In that case, *multicast* might be used. A packet which has a *multicast* IP address as destination will be copied and sent to all hosts which have registered to the multicast group represented by that address. IGMP is used to manage the membership of hosts to multicast groups [46]. The operation is encoded in the message in the *type* field. Among others, hosts can announce their inclusion to a group with a *membership report*, or leave such a group with a *leave group* message. The latest version of this protocol is version 3 [47], but we noticed during our measurements that version 2 [46] is still widely used in the context of SHs.

While ICMP and IGMP are in practice transmitted as payloads over an IP header, they are considered as part of the Network layer, *i.e.* layer 3, as they provide functionalities auxiliary to IP.

Address Resolution Protocol (ARP) The Network layer allows addressing packets at a global scale by using the destination host's IP address. How-

¹This generally depends on the physical medium used to transmit the packets. For example, IP over Avian Carriers [42] might suffer from frequent packet losses.

ever, at the lower Data Link layer, frames are transmitted from one host to the other by using their MAC addresses [41]. When a service only knows IP address of the host it must transmit the frame to, it can use the ARP [48] to resolve it to its MAC address. To do so, it issues an ARP *request*, which contains the IP and MAC address of the sender, as well as the target IP address to resolve. A neighboring host knowing the MAC address associated to the requested IP address answers with an ARP *reply* containing it.

2.1.2.2 Transport layer

While the Network layer is used to deliver a packet to the correct host, multiple services might run in parallel on a single host [37]. The Transport layer adds a *port* to a packet, which is an integer identifying which service issued a packet (the *source* port), and to which service it is destined (the *destination* port). Well-known services have standardized ports, which are identical across all hosts using them: HTTP runs on TCP port 80, DNS runs on UDP port 53, etc.

User Datagram Protocol (UDP) UDP [49] is a basic transport protocol. It only provides the default functionality of the Transport layer, namely identifying the services exchanging packets. It does not ensure a reliable delivery of the packets. As such, it benefits from a low overhead, and is recommended for services which require low latency, and for which sporadic packet losses are not critical, e.g. voice, video streaming, or online gaming.

Transport Control Protocol (TCP) TCP [38], like UDP, identifies the communicating services with ports, but also provides reliable delivery of packets. As such, TCP's operation is more complex than UDP's. First, before exchanging data, TCP endpoints must explicitly initiate a connection, by negotiating the options which will be used. Then, TCP ensures reliable packet delivery by leveraging acknowledgments for received packets, and retransmits the ones which have not been acknowledged. This complexity introduces a higher latency, but guarantees that the data will be delivered.

2.1.2.3 Application layer

The top-most layer, namely the Application layer is the one which finally provides delivery of the actual data [37]. Theoretically, any data encapsulated in a packet, higher than all the other layer, is considered as an application-layer protocol, so the number of potential application-layer protocols is endless. Nevertheless, common use cases have fueled the development of standard

application protocols, such as HTTP(s) to browse the World Wide Web. Four of them, which are often used in the IoT context, will be described hereunder.

Dynamic Host Configuration Protocol (DHCP) While any networking hardware usually has a fixed MAC address associated to it by the manufacturer, the IP address depends on the network the device is in. To alleviate the burden of manual IP address configuration, DHCP [50] was standardized. It operates as a client/server protocol. It allows a new device (the client) to query the DHCP server (the device acting as the authority to distribute IP addresses, usually the network gateway) to obtain an IP address to be able to communicate using layer-3 protocols. The DHCP process involves four different types of messages:

- DISCOVER: The client broadcasts that it necessitates an IP address
- OFFER: The server proposes an available IP address to the client
- REQUEST: The client requests the offered IP address explicitly
- ACK: The server acknowledges the request.

After this process, the client may use the offered IP address. DHCP runs over UDP: the server uses port 67, and the client port 68.

Network Time Protocol (NTP) The incentive behind NTP [51] is straightforward: providing a simple way for devices to retrieve the current time. It is implemented as a client/server protocol. A client which queries a server via a NTP packet will have its clock synchronized with the server (notwithstanding a slight skew of a few milliseconds). An NTP server listens on UDP port 123.

Domain Name System (DNS) As stated before, (public) IP addresses are able to address a host anywhere on the planet. This enables anyone to access a public service from anywhere; for instance, as I was writing these lines, the website `google.com` was reachable at the IPv4 address `173.194.76.101`. However, directly using IP addresses to reach services poses a couple issues. The most straightforward is that IP addresses are tedious to remember, and would therefore have to be stored somewhere. Additionally, due to how data centers operate, the IP address associated to a service might change regularly [52, 53]. DNS [54] was introduced to overcome those issues. Instead of referring to a service by its IP address, one can use a *domain name* (e.g. `google.com`), way easier to remember. Behind the scenes, though, it is still the actual IP address which is used to communicate. DNS provides the interface between both: it converts a domain name to its associated IP address(es).

In practice, a DNS client issues a query towards a DNS server (or resolver) containing the domain name to convert, and the server responds with the IP address. The initial specification of DNS states that the server should listen on UDP port 53; this is the case we encountered in all our experiments.

A variant of DNS, dubbed Multicast Domain Name System (mDNS) [55], is often used in the SH context. It applies the concepts of DNS, namely resolving a hostname to an IP address, but does so in a local network by leveraging multicast, and therefore does not necessitate a DNS server. This protocol is often used for automatic service discovery [56]: a device can query several devices at once for a specific service, and announce the services that it proposes. This is useful for plug & play installations, as the devices can discover automatically the available services around them. Devices wishing to be forwarded mDNS messages must register to the mDNS multicast group². mDNS messages are encapsulated in UDP datagrams; the source and destination ports are usually both set to 5353 (as a nod to DNS's 53).

Hypertext Transfer Protocol (HTTP) HTTP [57] is probably the most well-known communication protocol, even among people without any knowledge in computing, as it is the one we all use to browse the web. Indeed, web pages are transmitted via HTTP, as Hypertext Markup Language (HTML) documents. HTTP again operates in a request/response fashion, with a client/server architecture. The server listens on TCP port 80. To access a specific web page, the client must input its Uniform Resource Identifier (URI) [58], which contains, among others, the server's domain name or IP address, and the path of the page within the server. Different *methods* can be used, to perform different actions with the web page: GET simply queries the page, PUT uploads data, etc.

By default, HTTP transfers documents as plain text, without any authentication or encryption. Now that people use the web for many everyday actions, including sensitive ones (e.g. managing their bank account), this represented a significant security issue. The use of Transport Layer Security (TLS) [59] to secure HTTP communication has been standardized, forming the Hypertext Transfer Protocol Secure (HTTPS) protocol [60]. This new version of the protocol uses TCP port 443, to allow coexistence of both the secure and unsecured version.

Simple Service Discovery Protocol (SSDP) Finally, SSDP [61] has a similar incentive as mDNS, namely the automatic discovery of services in the local network. Its operation is comparable to mDNS: hosts register to the corre-

²IPv4 address 224.0.0.251; IPv6 address ff02::fb.

sponding multicast group³, then can announce and query services. Nonetheless, the data format is analogous HTTP, as SSDP transmits HTTP-like text-based documents, and uses methods to indicate the operation, *e.g.* M-SEARCH for discovery, and NOTIFY to advertise a service. SSDP uses UDP port 1900.

2.1.3 IoT-specific protocols

IoT devices that do not have resource or energy constraints, often because they are mains-powered, such as smart TVs, can employ the same communication protocols as general-purpose devices, *e.g.* the ones presented in Section 2.1.2. However, devices with limited computing capabilities and/or energy constraints, because they are, for instance, battery-powered, might need specifically designed protocols, which would be less complex, and therefore consume less energy [29]. In the following, we will present several of such protocols, sorted by increasing level of abstraction of the OSI layers.

2.1.3.1 Physical and Data Link layers

Regarding general-purpose devices, wired ones usually use Ethernet as Data Link-layer protocol, whilst wireless ones use Wireless Fidelity (Wi-Fi) [41]. However, those are not tailored for the IoT. As Ethernet is a wired protocol, it restricts the location and mobility. Wi-Fi, being wireless, mostly solves those problems, though its range can be limiting depending on the use case, and it is quite hungry in terms of energy consumption [62]. To minimize the energy consumed by IoT communication, specific protocols are often designed by tightly coupling the theoretical functionalities of two layers adjacent in the OSI model. Here, we present four wireless protocols which jointly define physical and data link layer functionalities. They all rely on the unlicensed, free frequency band dubbed the Industrial, Scientific and Medical (ISM) band, which operates at varying frequencies depending on the geographic zone (*e.g.* 868 MHz and 2.4 GHz in Europe) [63].

Bluetooth Bluetooth (BT) [64] is a short-range Radio Frequency (RF) communication technology. The Bluetooth link control hardware, integrated as either one chip or a radio module and a baseband module, implements the RF, baseband, and link manager portions of the Bluetooth specification. Bluetooth devices are generally organized into groups of two to eight devices called piconets, consisting of a single master device and one or more slave devices. The Bluetooth Low Energy (BLE) variant has been designed to consume less power, and is thus more adapted and optimized for IoT networks [65].

³IPv4 address 239.255.255.250; IPv6 site-local address ff05::c.

IEEE 802.15.4 Covering the two first OSI communication layers, *i.e.* the physical and data link layer, the 802.15.4 standard [66], proposed by the IEEE, was developed in 2003 to build low-power and low-rate wireless Personal Area Networks (PANs). It provides a communication range between 10 and 100 meters, and is therefore fit for Smart Homes or WSN where the sensors are located close to each other, but requires a multi-hop topology if long range communication is needed. As the standard only defines the first two layers of the OSI model, additional protocols have been designed to implement the upper layers, *e.g.* Zigbee and Thread (described in Section 2.1.3.2).

LoRa & Sigfox Low-Power Wide-Area Network (LP-WAN) technologies [67, 68] allow to build IoT network with point-to-point communication ranges up to several kilometers [69], and thus are particularly suitable for Smart City or agriculture applications. The two most popular LP-WANs are LoRaWAN [70] and Sigfox [71]. The former uses a proprietary physical layer modulation, LoRa, above which the free LoRaWAN data link layer is defined for Access Control List (ACL). In contrast, the Sigfox physical modulation is licensed to multiple manufacturers, but the MAC layer is proprietary. Other differences include the data rate, which is higher with LoRaWAN, and the range, which is greater with Sigfox [72]. As both technologies use non-IP stacks, they necessitate a gateway between the LP-WAN network and the Internet.

Z-Wave Z-Wave [73] is a proprietary protocol which describes full-stack communication functionality for home automation solutions. At the physical layer, it operates over one of the ISM bands (*e.g.* the 868 MHz band in Europe). It has a range of up to 100 meters for point-to-point communication and also supports larger mesh networks. It is based on a non-IP protocol stack; IP hosts must therefore communicate to Z-Wave devices via a Z-Wave-over-IP gateway that understands encapsulated Z-Wave messages.

2.1.3.2 Network layer

Routing Protocol for Low-Power and Lossy Networks (RPL) RPL is a routing protocol designed for IEEE 802.15.4-based mesh sensor networks and other unreliable networks [74]. In such networks, link failures are frequent and communication is expensive. RPL organizes nodes as a tree-shaped topology, specifically as a Destination-Oriented Directed Acyclic Graph (DODAG), optimized for the communication of IoT devices with external servers.

IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) The specification for 6LoWPAN defines encapsulation and discovery mechanisms allowing to operate IPv6 over IEEE 802.15.4 and other lower-layer

protocols [75]. To do so, it compresses the large IPv6 header by assuming default values for header fields unless explicitly stated otherwise, and using lightweight encoding for typical values. Since existing Internet transport and application-layer protocols can directly run over 6LoWPAN, the gateway only needs to translate 6LoWPAN packets to full IPv6 packets and vice versa to provide interoperability between the IoT network and the Internet.

Zigbee Zigbee is an IEEE 802.15.4-based specification by the Zigbee Alliance for the operation of wireless PAN [31]. It supports star, tree, and mesh network topologies. The nodes can be configured dynamically and have the capability of ad-hoc routing. A designated coordinator node manages the network and is also responsible for the generation of the network encryption key. The original Zigbee design has limited interoperability with IP networks, but a specification published in 2013, dubbed Zigbee IP [76], uses 6LoWPAN and RPL to route standard IPv6 traffic.

Thread The recent Thread protocol [77] has been tailored for next-generation IoT networks. At the physical and data link layers, it leverages the IEEE 802.15.4 protocol (similarly to Zigbee), which consumes less energy than more generic ones (e.g. Wi-Fi). Seemingly taking inspiration on the aforementioned Zigbee-IP [76], it provides built-in support for IPv6 thanks to its use of 6LoWPAN, and therefore integrates seamlessly with existing IPv6 networks. As such, it is able to transport segments which use typical protocols from the TCP/IP stack. Additionally, it is one of the preferred protocols to support the novel application-layer protocol Matter [78], described in Section 2.1.3.3, which intends to ease interoperability in Smart Homes.

2.1.3.3 Application layer

Message Queuing Telemetry Transport (MQTT) The MQTT protocol [79] is an application-layer messaging protocol following the *publish/subscribe* paradigm. Unlike the other protocols presented in this section, it is not specific to IoT, and usually runs over TCP, but has nevertheless found many usages in IoT applications [80]. It allows for large-scale asynchronous communication between publishers, that send messages related to a topic (e.g. a temperature sensor broadcasting a temperature update), and subscribers, that receive all the messages related to the topic they are subscribed to (e.g. a heater that must switch on or off based on the measured temperature). Between the publishers and subscribers lies a broker, which stores the topic subscriptions, receives published messages and forwards them to the subscribers [81]. The MQTT protocol is lightweight and open, which makes it suitable

for a wide range of applications, including in constrained environments, such as IoT and other Machine-to-Machine (M2M) communications.

Constrained Application Protocol (CoAP) CoAP [82] is an application-layer protocol which provides a REST-based request/response interaction model that is similar to HTTP and uses the same methods. However, the messages are specifically designed for communication with resource constrained devices over low-bandwidth networks. Since CoAP runs on top of UDP, it defines its own mechanisms for reliable communication, via four different message types: CON (confirmable), NON (non-confirmable), ACK (acknowledgement) and RST (reset). It can use Datagram Transport Layer Security (DTLS) for optional encryption, but due to the bandwidth and resource constraints, it can only be used with unicast messages. Although it is a pull-based protocol by design, an extension allows devices to periodically push measurements to the client, thus avoiding the overhead of repeated requests.

Modbus Modbus is an application-layer protocol created by Schneider Electric, initially developed for the communication between field devices (*e.g.* Programmable Logic Controllers (PLCs)) and control servers (*e.g.* Master Terminal Units) in ICSes [83]. As it is an open protocol easy to implement, Modbus is a *de facto* standard for ICS communication, such as in Supervisory Control and Data Acquisition (SCADA) systems. Given its prevalence in ICSes, and thanks to one of its variants running on top of TCP [84] which makes it possible to run on any device with IP capabilities, the Modbus protocol is a first choice for Industrial Internet of Things (IIoT) networks [85, 86, 87]. Modbus is a client/server protocol working in a request/response fashion where the field devices typically act as Modbus servers while the control host acts as a client [88]. The field devices are connected to sensors and actuators through Input / Output (I/O) modules. The state of the I/O ports of a device as well as its internal state are accessible through *registers* that can be queried or modified by the client by sending Modbus requests.

Matter Matter [78] is an effort, conducted by market-leading companies in the tech sector (*e.g.* Apple, Google, Cisco), to homogenize the communication between SH devices, regardless of the manufacturer. As a nod to its incentive, it was initially dubbed as project Connected Home over IP (CHIP). It operates at the application layer, and leverages IPv6 as network layer. The lower layers can be provided either by Ethernet, Wi-Fi, or the aforementioned Thread protocol (Section 2.1.3.2) To allow for its advertised interoperability, it defines an abstract data model, which must be followed by the devices implementing the protocol to be able to communicate with each other.

2.1.3.4 Others

Many other protocols and network technologies have been used to implement IoT networks, as presented in the survey by Al-Fuqaha *et al.* [56]. In particular, the usage of mobile networks has a long history, be it with the now obsolete 2G, the current LTE-M and NB-IoT (both 4G) [89] and the latest 5G [90]. We consider those protocols are out of scope, for the survey of IoT datasets presented in Chapter 3, as well as for both contributions in Chapters 5 and 6. For the former, we will detail the reasons in the related chapter. For the latter, we focused on the traffic occurring inside the Smart Home itself, *i.e.* wired and wireless traffic in the home Local Area Network (LAN), which does not cover cellular traffic.

2.1.4 IoT subdomains

By itself, the IoT is not really a concrete topic. Rather, it is a term which encompasses various sub-topics (which we will dub *subdomains*) with diverging practical applications, but which all share the characteristics which define the IoT [91], as highlighted in Section 2.1. The most popular subdomains will be presented in this section:

- Wireless Sensor Networks (WSNs)
- Smart Cities
- Connected Health
- Industrial Control

The subdomain of the Smart Home, as the focal point of this thesis, will be described in more detail in Section 2.3.

Wireless Sensor Networks

WSNs are quite self-explanatory: they consist of a multitude of simple devices, which have the single function of sensing their environment [92]. The actual quantities they are configured to measure is specific to the use-case; a widespread example is environmental data, such as temperature or air quality. Whilst individual devices bear little functionality, the interest of such networks lies in the sheer number of devices, allowing data to travel over long distances: any individual device can transmit its own data, as well as forward data from a neighbor.

Smart Cities

Smart Cities [93] take the concepts implemented to Smart Homes, namely the automation of various parts of the environment, and extend them to (parts of) an urban area. While the incentive is similar, the requirements are inherently different: communication must be performed over longer ranges. Therefore, other protocols are used, *e.g.* LP-WANs like LoRaWAN (2.1.3.1), or cellular communication with 5G onwards. Applications include smart traffic, city lighting, waste management, etc. Autonomous vehicles can be considered a specific case of the smart city domain: they must be able to communicate with each other, potentially over long distances, and add the requirement of mobility.

Connected Health

The health sector can benefit a lot from new capabilities brought with IoT technologies, and substantial effort has been conducted by companies and academia in this direction [94]. As such, the "health IoT" is often considered as a prominent subtopic of IoT. It is quite a transversal topic, intersecting with other domains, IoT or not, encompassing devices belonging to other categories, namely domestic connected health-monitoring devices, including wearables, which fall into the Smart Home category as well.

Industrial Control

Industrial sites have been controlled with *ad hoc* computing devices for a while, forming what is known as ICSes and SCADA systems. Those devices are historically very specific to their purpose, rendering interaction with other, differently purposed devices cumbersome. To tackle this issue, the concepts of IoT are gradually embedded into such industrial networks [95]: more generic and widespread devices and communication protocols are used, (*e.g.* the TCP-based implementation of Modbus [84]) streamlining data management and control over the industrial pipeline. The application of IoT concepts to industrial environments has been dubbed the IIoT.

2.2 Network security

The security of IoT devices is notoriously worse than that of typical, general-purpose hosts, such as computers and smartphones [96, 97, 98]. The most well-known examples of this bad security are the 2016 DDoS attacks launched by the Mirai botnet [5], which reached a traffic rate of up to one Tbps, the biggest volume for a DDoS attack at the time. As a botnet malware, Mirai infected vulnerable devices, turning them into "bots" which could then be con-

trolled remotely by the attacker, allowing them to perform arbitrary actions with the devices; in this case, issuing synchronized traffic towards Internet hosts to render them unusable. The malware's operation simply consists in a dictionary attack: it tries logging in to a Linux device by trying credentials from a list of frequent ones. Infection was successful on IoT devices, mainly home cameras, because those were left with default credentials. This textbook case demonstrates the unsatisfactory state, and yet tremendous importance, of IoT security. In the following, we will give clues on why such a lack of security (2.2.1), and highlight existing strategies to alleviate it (2.2.2).

2.2.1 Vulnerabilities

The relative vulnerability of IoT devices stems from a combination of several complementary factors; some will be explained below.

Limited hardware The reason for manufacturers to propose IoT devices on the market is of course to make profit, by selling a maximum quantity of devices. To do so, they want to entice customers by making the device as affordable as possible. One strategy is to purposely limit the hardware, and thus the computing power and memory, to reduce production costs. Consequently, this limits the functionalities that are possible to embark the device with. Again in an economical effort to stay market-competitive, manufacturers will often choose to fill up the available resources with functionality, neglecting transversal aspects such as interoperability, robustness, and, most importantly, security [96]. The latter is therefore often very basic, even non-existent; in practice, this is translated in weak cryptography, usage of outdated, vulnerable protocols [99, 100, 98], or default login credentials which are impossible to change, or even no credentials at all. Besides, such devices are usually not equipped with security software *e.g.* an antimalware.

Default configuration A couple of IoT categories are expected to be used by laypeople which do not necessarily possess technical knowledge related to the operation of such a device, *e.g.* Smart Home (see Section 2.3) and Connected Health (see Section 2.1.4). In such cases, users acquire IoT equipment for convenience. They assume their operation to be simple, even plug-and-play; they do not want to delve into heavy configuration. The IoT devices are therefore often left with default, vulnerable configuration, including default, easy-to-guess credentials [101]. Those represent a backdoor for attackers to compromise the device, as illustrated by the introductory example of the Mirai botnet [5].

Install-and-forget Related to the previous point, users tend to configure the device once, and let it do its job without verifying its correct operation. This might keep it with an outdated firmware exhibiting unpatched vulnerabilities [100, 99]. This risk is made even higher by the fact that such devices are usually kept powered on at all times, connected to other devices and/or the Internet, therefore easily reachable by malicious actors.

No direct interface Finally, a reason which does not directly make the devices vulnerable, but prevents users and operators to easily notice an unexpected behavior, is the fact the IoT devices do not generally provide a direct interface to the user, *e.g.* a screen. To detect an attack, one cannot simply passively look at the device; they must actively perform operational maneuvers, *e.g.* inspecting the device's network traffic or logs. Consequently, an ongoing attack might stay covert for an extended period of time, worsening its potential damage [102, 103, 5].

2.2.2 Security systems

Fortunately, there exist solutions to alleviate the inherent vulnerabilities of IoT devices. A straightforward one consists in monitoring, and potentially blocking, the traffic issued by or to the vulnerable devices. Such a function can be performed by a firewall or an Intrusion Detection System (IDS); the characteristics of those will be described in the next sections. Beforehand, we will describe the concept of a binary classifier, under which both aforementioned systems can be categorized.

It should be noted that the simplest, yet most effective method to protect IoT devices from vulnerabilities, is to remove those vulnerabilities through careful configuration. Weak default credentials should be changed for strong, unique ones, [5, 104] and known backdoors should be patched [105] by keeping devices up-to-date. While those suggestions seem simplistic, applying them already has the potential of shaking off most of the existing vulnerabilities.

2.2.2.1 Binary classifiers

A binary classifier [106] is a system which, as its name suggests, issues a binary verdict (yes/no, true/false, etc.) over the input it is given; the latter can take any arbitrary form. A concrete example would be a classifier which takes network packets as input, and, for each, decides to let it pass or block it. This specific classifier is called a *firewall*; its operation will be detailed in Section 2.2.2.5.

The correctness of a binary classifier is tied to its ability to correctly flag positive or negative input as such. As there are two input categories and two

Actual label	Positive	Negative
	True Positives (TP)	False Negatives (FN)
Negative	False Positives (FP)	True Negatives (TN)
Predicted label		

Figure 2.4: Sample confusion matrix

possible verdicts, there are four possible outcomes: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Those are usually summarized in a *confusion matrix*, as depicted on Figure 2.4.

2.2.2.2 Position

Inside a network, multiple devices can host a security system. We will refer to the actual device doing so as the “position” of said system. Its position influences the system’s decision capability, as it offers a compromise between availability of data and range of action.

Host-based A system which is deployed directly on the end devices (computers, servers, IoT devices, etc.) is dubbed as *host-based* [107, 108]. Such a system benefits from an extended knowledge of the device’s internal operation, and has therefore access to metrics such as resource consumption (memory, disk, I/O, etc.). On the flip side, its vision of the network is limited to the traffic which reaches the device on which it runs; it lacks a global view of the network. Its range of action is limited to the device it is attached to. Regarding scaling, when the number of devices increases, the number of instances of the system must jointly increase, while the resources allowed to one instance may stay low.

Network-based A *network-based* system [109, 110] is deployed in a central position in the network, *i.e.* on a network gateway or Access Point (AP). As such, it benefits from a global view over the network, and can take decisions more insightful for the complete network, but it lacks access to internal device metrics. As several end devices are connected to it, it has the potential to

protect all of them at once, but that also makes it a single point of failure. As the number of devices in the network grows, the central device running the system would need to be provisioned with more resources to support the amount of data produced by all the devices it monitors.

2.2.2.3 Decision logic

Security systems can differ in the logic they apply, and the input they are given, to issue a verdict.

Signature-based A *signature-based* system [111, 112] takes a list of known malicious traces (of any format), *i.e.* the signatures, as input, and rejects any trace which showcases a similar behavior. The advantage of this logic is its precision: only traces that are certain to be malicious are flagged. However, it fails at detecting new, unknown attacks, as a signature does not yet exist for those, which means it can suffer from a high number of false negatives [113, 108]. In the context of firewalls, this strategy is implemented as a *deny-list*, which is a configuration explicitly specifying the packets to block, while any unspecified one is accepted.

Anomaly-based Conversely, the input to an *anomaly-based* system is a description of the device's or environment's intended operation; a so-called *model* or *profile*. Any trace which does not comply with the model, *i.e.* which exhibits an *anomaly*, is flagged as malicious [114, 115]. While this logic is able to detect unknown attacks, it also tends to flag legitimate operation, if the latter is unknown *a priori* and was not embedded into the model. It therefore suffers from a higher number of false positives [113, 108]. For a firewall, the opposite of a *deny-list* is an *allow-list*, which specifies only the packets to allow, while any other is blocked.

2.2.2.4 Usage of state

Security systems may or may not leverage an internal state as an additional insight when selecting their verdict.

Stateless For each given input, a *stateless* system applies its verdict only based on the input itself [116]. It does not consider any input which occurred in the past. Such a system has a lower memory consumption, as it does not need to store any state.

Stateful On the contrary, a *stateful* system keeps a state in memory, based on the input it received previously. It can then leverage this state to aid in its

verdict for a given input [116]. The state can potentially be updated for every piece of input received. Storing a state incurs a higher memory consumption, but is able to increase the decision capability of the system. For an input composed of network packets, an example of a basic state is to keep track, for a given remote host, if a TCP connection has been completed [117]. Plain TCP packets (not bearing the SYN flag) can be accepted for open connections, but rejected for hosts without an open connection.

2.2.2.5 Concrete security systems

After having described the abstract characteristics shared by all network security systems, we present two concrete families of systems implementing those concepts. They basically differ in terms of complexity and resource requirements: firewalls are more lightweight but lack some complex capabilities of IDSes.

Firewalls A firewall is a software tool which inspects packets and issues a verdict, either accepting or rejecting the traffic [118]. The decision is based on traffic-based attributes, potentially at every layer of the OSI model except the physical layer; the most used ones being the MAC and IP addresses, the protocol type and the transport-layer ports. A firewall's operation being relatively simple, it can be used as a proactive security method, blocking traffic as soon as an anomaly is detected. The most well-known firewall was *IPTables* [119] which was easily portable as it operated in the Linux kernel. As such, virtually any Linux host could run it, whether a general-purpose computer, or more specified devices such as a router running OpenWrt [120], a Linux-based embedded OS tailored for network equipment. *IPTables* is now deprecated, superseded and enhanced by *NFTables* [121], which streamlines existing functionalities while adding new ones.

Intrusion Detection Systems An IDS [122] can be seen as a more complex version of a firewall: in general, it boasts a more complex decision logic, and can work with a wider range of input formats: packets, such as a firewall, but also flows (aggregated packets), system call traces, or even any *ad hoc* logs. The drawback of such flexibility is that an IDS cannot be used as a proactive security method, as it requires a sufficient amount of input data to produce an accurate decision, even more so as many modern IDSes leverage Machine Learning (ML) or even Deep Learning (DL) models to issue their verdict [115, 123]. As such, they are often used *a posteriori*; the *detection* part of the name stems from this. Still, if an IDS is used inline during operation, and is able to block actions flagged as malicious, it is referred to as an Intrusion Prevention System (IPS) [124, 125], since it does not simply *detects*, but also *prevents*.

The most popular IDS/IPS is the open-source Snort [126]. It works by inspecting network traffic, either inline (acting as an IPS) or *a posteriori* (resp. IDS). Before the inspection, Snort is configured with a set of rules, which describe the traffic it should flag as malicious. As such, its operation is similar to that of a very expressive firewall, applying the signature-based strategy [111]. Suricata [127] is another popular IDS which operates similarly to Snort. Suricata's initial edge over Snort is that it was designed as a multi-thread tool from the get go, which allows for a better performance than Snort [128].

Yet another popular open-source IDS is Zeek [129]. It has a similar usage as Snort in that it analyzes network traffic, but cannot work by itself as an IPS. On the other hand, it is more flexible than Snort in that it can act as a signature-based IDS [111] but also as an anomaly-based one [114].

2.2.2.6 Choice of security features

In the context of this thesis, we designed and implemented a firewall for SH networks, which is described in Chapter 5. For each of the aforementioned features, we had to pick the variant which is the most relevant for our use case. In the following, we summarize them, and give a quick rationale. The firewall is described in more detail in the corresponding chapter.

Position As our firewall is intended for a SH environment, we designed it as a *network-based* security system. It was deployed on a LAN AP, a device located at a central position in the SH network. Section 2.3.2 explains in more detail why, in the SH case, a *network-based* system is often more suited than a *host-based* one.

Decision logic The main purpose of our work in Chapter 5 is to design a format able to model the expected traffic issued by a SH device. This approach fits perfectly with the *anomaly-based* decision logic [114], as the model specifies what traffic should be accepted; any other traffic should be rejected.

Usage of state Our format allow expressing sequential dependencies between network patterns. For our firewall to take that into account, it must remember which patterns already occurred, by keeping track of a state. Therefore, our firewall is *stateful* [116].

2.3 Smart Home

In the context of this thesis, we focused on a specific subdomain of the IoT, namely, the Smart Home (SH) [130, 131, 132]. Its concept is to augment household everyday objects, from the smallest power plugs and light bulbs to big

appliances such as a washing machine or a fridge, with IoT functionalities (as discussed in Section 2.1). First, the capability to sense their environment (e.g. room temperature), as well as metrics related to their own operation (e.g. battery level, energy consumption). Second, the ability to communicate over a network.

The main purpose of this concept is to enable *home automation*, which consists in automating any interaction which could take place in a household [133, 134, 135]. Here are various examples of such:

- When a sensor detects motion in a room, the light in the same room turns on.
- At night, the heating and all lights turn off.
- When the door lock and a motion sensor in the corridor are activated conjointly, it means the inhabitant is coming home; this triggers the heating.

Such interactions are often called *routines* [136, 137].

The ultimate motivation behind those routines is bringing more convenience to the Smart Home user. Indeed, they allow them to trigger a complex series of actions by pressing a single button, or even without any action at all. Paradoxically, they also bring more possibilities of failure in the environment: if only one of the devices in the interaction fails, it causes cascading failures of all interactions leveraging it. Doubled effort should be undertaken to ensure device failures occur as rarely as possible. In Chapter 6, we will assess if devices are indeed designed to be robust, and are able to ensure the success of a user-triggered action by providing backup communication patterns which make up for an unsuccessful one.

This section will delve deeper into SH-related concepts, which are central to this work, and which will be used as a base to describe the contributions in the following chapters.

2.3.1 Smart Home network topology

A SH network, as its name suggests, takes place in its users' home. In practice, this means that it will take the form of a LAN containing the SH devices [130, 138], and potentially other general-purpose devices, most likely the inhabitants' personal computers and smartphones [139]. In a domestic LAN, the central point of the network is the Customer Premises Equipment (CPE), a router provided by the Internet Service Provider (ISP) which provides Internet access [140]. This makes most SH networks adopt a star topology, as described in Section 2.1.1.3, with the CPE being the center of the star. Figure 2.5 depicts an example of a basic SH network

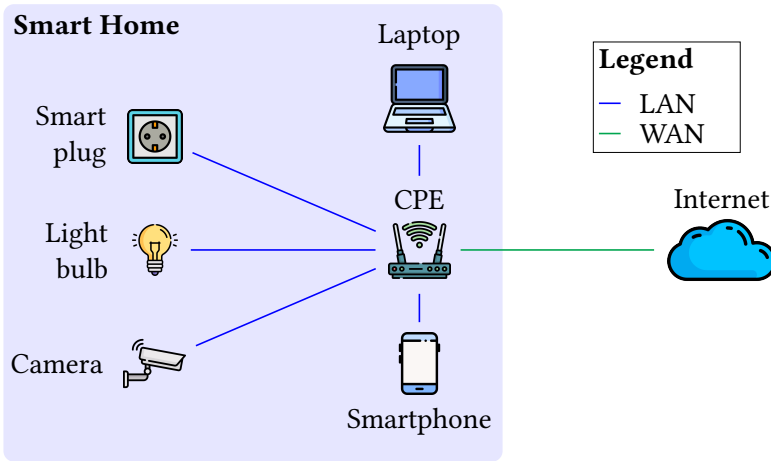


Figure 2.5: Example Smart Home network

The user might decide to include all of their devices in the same LAN, or, on the contrary, to partition the network into different Virtual Local Area Networks (VLANs) [41], with, for example, one for the general-purpose devices, and one for the SH ones [138]. The second solution is depicted on Figure 2.6. It is usually easy to set up, and provides a structural measure helping security and privacy. Indeed, as the SH devices are considered in a different LAN than the general-purpose ones, they cannot see which devices are in the other VLAN, and cannot contact them directly.

Admittedly, this can also prevent some home automation tasks, or at least render them more difficult or convoluted, with traffic having to go through the cloud instead of staying in the LAN. A usual interaction in a Smart Home is simply the user using the companion app on their smartphone to instruct a device to perform an action, *i.e.* the smart plug to toggle on/off. If the smartphone and the smart plug are in the same LAN, they might communicate directly with each other, without using the cloud as an intermediary. However, if they are in different VLANs, this direct communication is not possible, and the cloud intermediary is probably required.

2.3.2 Security in Smart Home networks

For now, we have described network security systems in a generic way. In this last section, we will specify the aforementioned concepts to the case of SH networks.

General-purpose computers tend to run an antimalware, either installed by the administrator, or even built-in such as Microsoft Defender on Windows computers [141]. This is often impossible on IoT devices, including SH

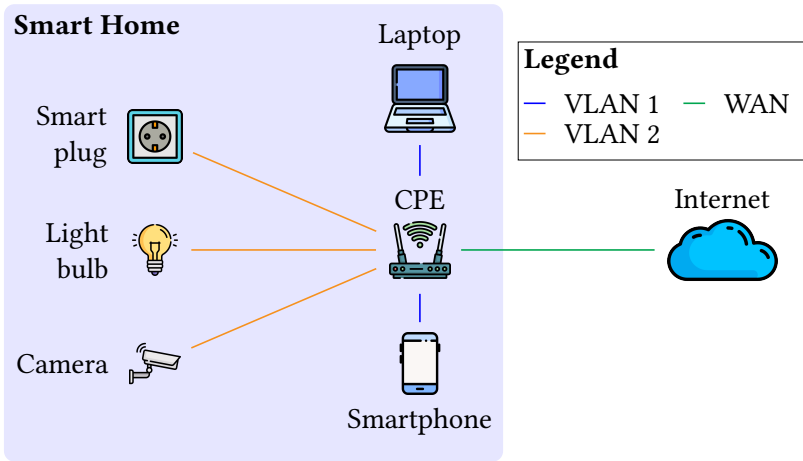


Figure 2.6: Example Smart Home network with two VLANs

ones, for two reasons. First, as stated in Section 2.2.1, IoT devices, including SH ones, are shipped with a limited hardware, which does not provide the sufficient resources to run a full-fledged security software [96]. Besides, even if the available resources would theoretically support running such software, IoT devices are usually way more rigid than general-purpose devices. In general, they do not allow the user to install any other module to the device, including a potential one for security [142].

Figure 2.7 illustrates the deployment of a host-based security system in a SH environment. In practice, there are two possibilities to do so, which both suffer from a substantial drawback. First, in the rare case that the device’s firmware is actually sufficiently open to run custom software, (e.g. a Linux-based device or one running an open-source firmware such as Tasmota [143]), one may implement a host-based tool directly on the device [144]. However, as the device provides only limited computational resources, the system must stay lightweight, which potentially compromises its efficacy. Then, in the common case that the device’s firmware is closed, a host-based deployment would require a proxy for each end device [145]. The latter would forward the necessary data to the former, enabling it to perform the security logic. As the proxy is a standalone device, it can be provisioned with sufficient hardware, removing the resource problematic. However, this also considerably increases the cost of the solution.

Due to the drawbacks of a host-based security solution in a SH setting, network-based solutions are more frequently considered [146, 147, 115, 24, 148]. Instead of installing an instance of the security system on each end device, a single instance is placed on a device central to the network, namely

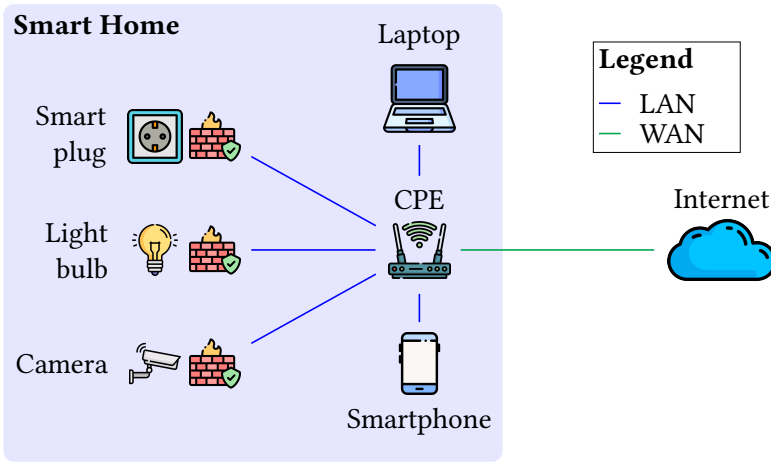


Figure 2.7: Example Smart Home network with host-based security

an AP (which can be the network gateway). In terms of capabilities, the CPE provided by the ISP, which is a router acting as the network gateway, is generally similar to the SH end devices themselves: limited in hardware [149, 150], and closed, *i.e.* impossible to run a module directly on it. The problem of requiring an additional device with enough resources and the possibility to run custom software still stands. However, deploying a network-based system only requires one additional device, compared to one per end device in the host-based case.

Figure 2.8 shows two possibilities in how to deploy this additional device. As depicted on Figure 2.8a, it can be installed on the path of the traffic [151, 24, 115] *i.e.* as a LAN AP, with the SH devices connected to its LAN ports, and its Wide Area Network (WAN) port connected to the network gateway, *i.e.* the CPE. In this case, the system can directly act on live traffic, potentially blocking it if an attack is detected. However, it also acts as a bottleneck on the traffic, introducing latency if the security operations take a long time. This approach is the one we followed for the implementation of our firewall system described in Chapter 5.

Alternatively, as illustrated on Figure 2.8b, it can be positioned outside the traffic's path, and being given a copy of the traffic by port mirroring from the network gateway [147]. This solution is less intrusive and do not introduce latency, but cannot directly act on the traffic; it can only raise alerts if malicious traffic is detected.

A way to modify the latter solution to still allow it to operate on the traffic, is to augment the network with Software-Defined Networking (SDN) capabilities, and make it such that the security system can act as an SDN controller,

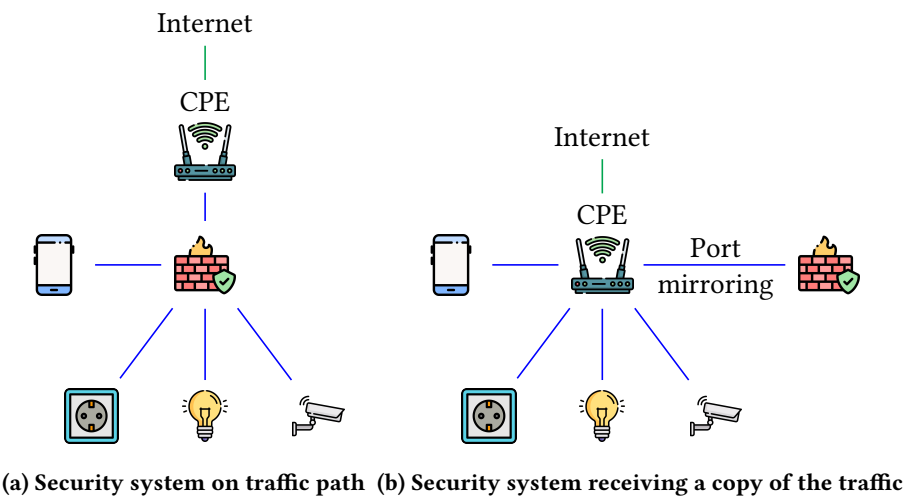


Figure 2.8: Example Smart Home network with network-based security

or interact with one [152, 148]. Such a controller can dynamically modify the traffic rules to allow or reject traffic based on the security logic. This hybrid solution shares the advantages of both previous ones, being able to block ongoing attacks while not introducing latency on the traffic.

Existing Public Datasets for Research in IoT Network Security

3

For any scientific discovery to be validated, it must undertake a sound and thorough evaluation. A paramount requirement for that is a good dataset, which serves as the base material for the analysis. However, such data might prove challenging to obtain, even more so for emerging research fields, what research IoT can still be considered today. Our first contribution in this thesis is a survey of existing public datasets covering various aspects of the research in the security of IoT networks as a whole.

When the research topic of IoT network security started gaining traction, no dataset tailored to this topic existed yet. To validate their findings, researchers tended to use datasets containing generic network traffic, which do not necessarily exhibit the same features as IoT-related traffic, whether in terms of protocols or traffic patterns. Some datasets used in that early era became popular in the community, and were still used at the time of this study, *e.g.* the NSL-KDD dataset [153], even if more adapted datasets had been produced. We deemed a comprehensive compilation of existing public datasets would be beneficial for the community. As the research presented in this thesis is inscribed in this topic, we also intend to identify the ones which could be potentially useful for the evaluation of our own contributions.

We focus on datasets which contain network traffic produced in IoT networks. Due to the variety of protocols used in such environments, we split the set of datasets in two parts: on the one hand, the ones containing traffic using protocols from the TCP/IP stack; on the other hand, those leveraging protocols designed specifically for the IoT, including those presented in Section 2.1.3. For the sake of exhaustiveness, we also included datasets which contain generic traffic not related to IoT networks, but which are still considered relevant for the evaluation of IoT-related works. Besides, we also included datasets containing data other than network traffic, *e.g.* system call traces, location, or physical metadata about the signal.

The work presented in this chapter has been published in an extended form as an article in the journal *IEEE Surveys & Tutorials* [1]. In the pub-

lished version, each covered dataset is described in detail, something that was omitted from this thesis for the sake of brevity.

3.1 Introduction

While the security of the IoT has become a prime concern, its implementation is quite challenging, for various reasons explained in Section 2.2.1. Due to the limited computing resources embarked in IoT devices, the most promising solution appears to be using a Network-based Intrusion Detection System (N-IDS), deployed on a device central to the network, such as the router acting as the network gateway, instead of on the end devices themselves [151, 115, 154]. Those can benefit from augmented resources, and have a view of all communication issued inside the IoT network. As such, they can monitor the network traffic arriving to or originating from IoT devices and classify it as benign and malicious, the latter triggering an alarm or a countermeasure. The state-of-the-art for such a system is to implement it as a binary ML classifier [115, 147, 155].

To train and validate an ML-based classifier, datasets are needed that contain a relevant subset of the data the classifier will be later applied on. In many cases, the definition of whether a classifier is “good” or not is meaningless without the support of a large amount of empirical data. In the case of an N-IDS, the classifier must be trained and validated on a sufficiently large, representative, and correctly labelled dataset consisting of benign and malicious network traffic.

In general, the relevance of a dataset for this task can be measured along three axes, namely *Quantity*, *Quality* and *Availability* [156]. The quantity refers to the size of a dataset, *i.e.*, the number of records it contains. The underlying expectation is that the larger a dataset is, the more likely it contains representative samples. By extension, quantity can also refer to the number of different datasets covering similar data. Quality represents the likeliness of the dataset to produce useful results when used, *i.e.*, to emulate real-world cases. For a dataset consisting of network traffic, it encompasses diverse characteristics, such as the number of different end hosts appearing in the dataset, the duration of the traffic collection, the constraints of the tools used for the collection process, etc. Datasets exhibiting low quality data may induce biased, or even wrong results. Finally, availability refers to the fact that researchers need access to the dataset. Obviously, releasing datasets for reproducibility is an important part of the scientific method [157].

Different approaches exist to obtain IoT-related network datasets. In *test-bed-driven* generation, researchers generate datasets by instrumenting an environment containing IoT devices and recording the network traffic under normal conditions, as well as under attacks. This is demanding work, as it

necessitates resources, in terms of money (*i.e.* to obtain hardware) and time (*i.e.* to perform the data capture). In contrast, *synthetic* dataset generation requires less resources as it is based on the simulation or emulation of the IoT devices, the communication network, and the applications running on top of it. This approach is very flexible, but its difficulty lies in achieving a realistic behavior of the different components. Finally, *empirical* datasets are obtained by recording the network traffic of real IoT devices used by real users. Such datasets are “realistic” by design, but collecting them comes with various technical and non-technical challenges, such as privacy concerns.

If researchers cannot collect their own dataset, they can employ a dataset that has been publicly released to the community. Finding such a dataset that fits one’s needs for a specific use case can be a tedious task. To help the community with this endeavor, we conducted a comprehensive survey of publicly available datasets related to IoT network security research, while classifying them along various characteristics. To this end, we analyzed over 100 scientific articles and more than 70 datasets. Not all datasets are suitable for network security research; therefore, we have limited the scope of the research as explained below.

3.1.1 Scope

We focused on datasets that can be used to design and validate network security solutions for IoT networks. Here, the term “IoT network” refers to the network allowing IoT devices to communicate with other devices and services, such as cloud servers or smartphones running companion apps. Most of the communication protocols covered have been described in Sections 2.1.2 and 2.1.3. If a specific protocol is involved in only one dataset, it will be described along said dataset. We are *not* interested in network communication between general-purpose devices, *i.e.* PCs, servers, smartphones or tablets, even if happening in the context of an IoT application, for example between a database server storing IoT sensor data and a web browser querying that data. The covered datasets mainly represent two IoT subdomains, as defined in Section 2.1.4: Smart Home, the one which is the focus of this thesis, and Wireless Sensor Network.

Cellular networks are widely used to provide communication boasting long range and good coverage nearly everywhere. Their most popular use case is to accommodate the communications of cell phones, with the series of “G” protocols, *e.g.* 4G or 5G. Variants offering a lower rate and therefore a lower energy consumption have been developed, intended for IoT solutions, namely LTE-M and NB-IoT. While the latter can be considered as IoT-specific protocols, we do not include them in our study. Actually, datasets containing IoT-related traffic based on cellular protocols are actually very scarce, even

nonexistent, at least at the time of the research. A couple of reasons might explain this. First, capturing cellular traffic is way more arduous than the other protocols presented. LAN traffic can be captured by simply attaching an AP, *e.g.* a Raspberry Pi, to the home router. On the other hand, cellular traffic requires specific hardware able to sniff the specific frequency bands. Besides, as cellular communication is ubiquitous and boasts a long range, sniffers might capture a large quantity of parasitic traffic, which should then be filtered out and/or anonymized. Nevertheless, for completeness, it should be noted that a recent study [158] collected a dataset of cellular traffic, including NB-IoT, even if the latter was not specifically the focus of their work.

In the context of Industry 4.0, the IoT paradigm has also gained popularity for industrial control and monitoring, forming the subset of IoT dubbed Industrial Internet of Things (IIoT). We also consider datasets from this domain as in scope for this paper. However, we do not include datasets gathered at traditional ICS and SCADA systems, such as [159]. While those systems also use connected sensors, they differ in several points from IoT networks, such as in the usage of PLCs, real-time capable hardware and software, and, in some cases, proprietary network protocols and equipment [160]. Similarly, we do not consider datasets containing the internal communication between Electronic Control Units (ECUs) or similar embedded systems, *e.g.* from the Controller Area Network (CAN) in cars, such as [161].

As this survey focuses on network traffic, datasets containing only the *content* of IoT communications, *e.g.* telemetry data, are *not* considered here. Such datasets exist and contain, for example, information gathered by sensors in parking lots, records of user interactions with SH devices, health data from smart health devices, or soil information collected by a WSN in a plantation. Similarly, host-centric digital forensics datasets, containing information such as disk images, computer malware binaries, or RAM dumps, are not covered.

It is worth noting that we have included several popular IP traffic datasets that do not contain any IoT traffic. We have done so because these datasets have been used by researchers working on IoT security, either because they did not have access to IoT-specific datasets or because IoT datasets with the desired content did not exist. The datasets in question may still be valuable to IoT researchers, and one could indeed argue that modern IoT devices using general-purpose Internet protocols such as HTTP might behave similarly to traditional Internet hosts.

Finally, for some of the datasets mentioned, we did not find a way to obtain them directly. Those have been nevertheless included for two reasons. First, they could potentially be obtained by directly contacting the authors. Second, we deemed those datasets contain relevant data or show an interesting measurement methodology which could be reproduced by researchers for similar use cases.

3.1.2 Contribution

In the following, we survey the state-of-the-art in datasets for research in IoT network security, by giving a clear view over the available datasets and their properties. For the latter, we have identified and described in detail 11 features, which characterize and the covered datasets, and might help researchers finding the most suited to their particular use case. In total, we present and classify 74 datasets, focusing on the ones that are related to IoT devices, include network traffic, preferably benign and malicious, and are publicly available.

Furthermore, we discuss issues with existing datasets and point to future directions in the creation of IoT-specific network datasets. Therefore, this work can also serve as a guide to identify protocols and scenarios neglected so far by dataset creators.

3.1.3 Structure

The rest of this chapter is organized as follows. In Section 3.2, we present related surveys and research. In Section 3.3, the methodology we followed to perform the survey and to select the relevant datasets is explained. Section 3.4 details the characteristics we defined for the classification of the datasets, whereas Section 3.5 summarizes the covered datasets themselves, along with their classification following the aforementioned characteristics. Section 3.6 discusses current challenges and future directions for the generation of IoT datasets. A conclusion for the chapter is drawn in Section 3.7.

3.2 Related Work

To our knowledge, there has been no previous extensive survey of network trace datasets for IoT security. However, there have been publications on other types of devices and other aspects of (IoT) security. In the following, we will present them and discuss their relationship to our work. Table 3.1 gives a summary.

Jing *et al.* [164] proposed a categorization of security-related measurement data. For each category of data, the authors provided a specific classification and discussion of its advantages and disadvantages. Four high-level categories of data were presented: packet-level data, flow-level data, connection-level data and host-level data. Packet-level data contains packet header information, packet payload information, and packet activity information. Port mirroring is one of the commonly used packet collection methods. Flow-level data contains information about *flows*, *i.e.* streams of packets that share one or more attributes. Flow-level measurements reduce the measurement overhead and can be useful if the packet payload is not needed.

Ref.	Year	Network	Datasets?	Description
[162]	2016	G.	✓	DM&ML methods for cyber analytics. General-purpose datasets for IDSes. Challenges in using ML/DM for cybersecurity.
[156]	2017	/	✓	Datasets for digital forensics, containing e-mails, RAM dumps, disk images, network traffic, etc.
[163]	2017	IoT	✓	Datasets for mobile, wearable and IoT research.
[164]	2018	G.	✗	Collection, analysis and requirements of security-related data. Methods for DDoS detection.
[165]	2019	IoT	✓	Classification of IoT security threats and challenges. ML-based N-IDS for IoT.
[166]	2019	G.	✓	Datasets for intrusion detection, classified along 15 features.
[167]	2021	ICS	✓	Architecture, devices and security protocols for ICS testbeds and datasets.
[168]	2021	G.	✓	Device types, behavioral data, and processing and evaluation techniques for device identification and malicious activity detection.
Ours [1]	2023	IoT	✓	Mostly public datasets containing IoT traffic traces (but also other kinds of data and networks), classified along 13 features.

Table 3.1: Related surveys (sorted by date)
G.: Generic (general-purpose) networks

Connection-based data consists of statistical information about the traffic between two IP addresses. Host-level data is collected on a host device and contains information describing its internal state and its evolution during an attack. The authors do not present any dataset.

Grajeda *et al.* [156] surveyed the available datasets for digital forensics, along with their origin and types. The datasets contain information that is important for the understanding of the behavior and impact of malicious software on a host device, such as RAM dumps, email datasets, apk files, etc. More than 50 datasets were presented, of which only three contained network traffic. Moreover, the authors discussed weaknesses in the existing datasets and future directions, such as lack of variety, data de-identification, and strategies to share complex data.

Chaabouni *et al.* [165] focused on IoT N-IDSSes based on ML. The authors explained the specific aspects of learning techniques for IoT security. The survey also presented several free datasets and N-IDS implementation tools for further research. However, since the survey's focus was on detection techniques, only six datasets were discussed.

Buczak *et al.* [162] presented ML and Data Mining (DM) methods for cyber analytics. They discussed in detail the different types of data used by ML and DM approaches (Packet Capture (PCAP) captures, Netflow, etc.) and mention four public datasets (DARPA 1998 [169] & 1999, KDD'99 [170], NSL-KDD [171]) containing network traffic from general-purpose computers. Most of these datasets are outdated.

Ring *et al.* [166] provided a survey of more than 30 datasets for IDS research and described the packet-based and flow-based data in detail. The authors proposed 15 different properties that cover a wide range of criteria to assess the existing datasets. Most of them are similar or identical to our own characteristics, *e.g.* year, public availability, presence of attack traffic, duration, etc. The ones that we do not include were deemed less important (*e.g.* anonymity, as payload encryption becomes more widespread), equal for nearly all datasets (*e.g.* presence of normal traffic), or linked to another characteristic (*e.g.* size of data, which is linked to the recording duration). The survey is comprehensive, but the datasets mostly consist of traffic from general-purpose computer networks and hosts.

Sánchez *et al.* [168] presented a review of different types of devices and state-of-the-art assessing techniques for device identification and misbehavior detection. The authors also discussed 37 public datasets of which 30 contain network data, of which, in turn, only 10 contain IoT traffic (one contains radio spectrum measurements from IoT devices). The other datasets contain network traffic from general-purpose computers and industrial control systems, as well as non-network data (telemetry data, host performance metrics, etc.).

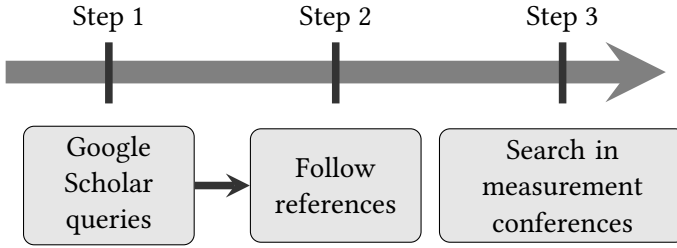


Figure 3.1: Methodology for datasets gathering

Conti *et al.* [167] proposed a comprehensive overview of Industrial Control Systems (ICS) along with their security challenges, and presented different ICS testbeds and datasets. They reviewed around 50 testbeds, classified along nine features, and over 20 datasets categorized by 12 characteristics. As we did with Ring *et al.* [166], we share some of those characteristics, namely the type of data, recording duration, and presence of attacks. Their characteristics that we did not share include metadata (*e.g.* dataset name, reference and resource), dataset format (PCAP, CSV, etc., that we detail in the text), percentage of attacks inside the dataset, or evaluation metrics (F1-score, accuracy and precision). We deemed the latter not relevant for a survey, as those values depend on the detection algorithm. The authors also detailed good practices for researchers who want to design a testbed. The paper discusses in detail ICS-specific aspects, such as commercial PLCs, HMI stations, and SCADA protocols, which are less useful in the IoT context.

Rawassizadeh *et al.* [163] surveyed dataset repositories, which are online platforms gathering publicly available datasets. They focused on repositories containing datasets related to mobile, wearable and IoT research. They listed four repositories, namely UCI Machine Learning Repository, UCR Time Series Archive, PhysioBank and CRAWDDAD. Other well-known repositories with more general datasets were also mentioned, such as KDnuggets, DRYAD, Datahub, etc. The authors did not detail the characteristics of the individual datasets found in those repositories.

3.3 Methodology

This section presents the methodology that we used to find the datasets for this survey. An overview is shown in Figure 3.1.

In the first step, we queried Google Scholar for articles with general keywords such as "IoT dataset". We selected the most recent ones, going from 2022 back to 2017. For this first step, we limited ourselves to 2017 for two reasons. Firstly, as computer science is a quickly evolving topic, datasets that are only a few years old might already be outdated with respect to current stan-

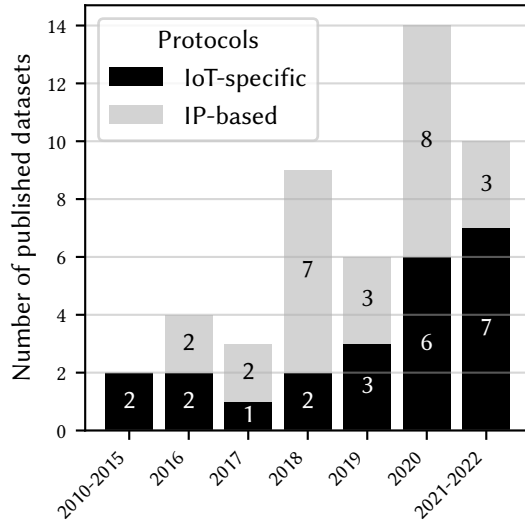


Figure 3.2: Number of published IoT-related datasets per year

dards and use cases. Secondly, since there are still two steps (see below) after the first one, yielding other datasets to cover, we had to set an arbitrary limit to not end up with an intractable amount of search results. We then made sure the selected articles were relevant by reading the abstract and skimming over the text.

As a second step, we looked for articles referenced by the articles we found in the first step. This led us to older datasets that were used, for example, by the authors as a starting point for their own creations, and to less cited articles that were relevant, but did not appear in the first results of a Google Scholar query.

Finally, as a last step, we explicitly searched for publications in three renowned network measurement conferences, namely the Internet Measurement Conference (IMC) [172], the Passive and Active Measurement (PAM) conference [173], and the Network Traffic Measurement and Analysis (TMA) conference [174]. This allowed us to gather a last set of datasets that did not appear in the first and second steps.

During the above steps, we also found several dataset repositories, *i.e.* collections of datasets. Those will be listed as a standalone dataset, but major datasets hosted in such a repository will also be listed individually.

Figure 3.2 shows the number of published datasets containing IoT traffic, split into two categories: the ones that cover IoT-specific protocols, and the ones that cover only classic IP-based protocols. We can see an increasing trend over time in the total number of datasets. Yet, their absolute number is



Figure 3.3: Feature-Oriented Domain Analysis: legend

still underwhelming, compared to the usage explosion of IoT in the last years. The ratio between IP-based and IoT-specific protocols is quite balanced: in some years, the former is larger, whereas in some others, the latter is. However, the aggregate category of IoT-specific protocols actually includes various protocols, which means that some are studied very little, or not at all. We further discuss this in Section 3.6.1.

3.4 Dataset Characteristics

This section will present the characteristics defined and used to classify the surveyed datasets along different axes. In the forthcoming sections, those characteristics will be evaluated on each dataset to highlight their strengths and weaknesses and identify their potential use cases.

Those characteristics have been structured following the principles of Feature-Oriented Domain Analysis (FODA) [175, 176]. Basically, we order them in a hierarchical way, and the resulting complete classification is depicted as a tree. Nodes which represent a standalone feature may be labeled either as *mandatory* or *optional*; the names are self-explanatory. In our tree, an optional feature represents a *boolean* variable: if its value is *true*, it will be present in the tree, whilst if its value is *false*, it will be absent. If a feature represent a categorical value, one may add the possible categories as children of the feature. The choice of the value is then represented either as an (inclusive) *or* relationship, or as an *alternative*, which means an exclusive or. The four aforementioned concepts are visually represented on a FODA tree; the associated legend is shown on Figure 3.3.

For visualization, as our tree is quite large, we split it into multiple figures. Figure 3.4 shows the first level of classification, as well as the children of one feature, namely the *Scale*. Other first-level features which have children are represented in the subsequent figures, *i.e.* Figures 3.5, 3.6, 3.7, 3.8, and 3.9.

3.4.1 Traffic type, data content, presence of attacks

The following three characteristics are used to describe the content of the dataset.

The *Traffic type* indicates whether the dataset has been created by using general-purpose devices (*e.g.* PCs and smartphones) or IoT devices. When

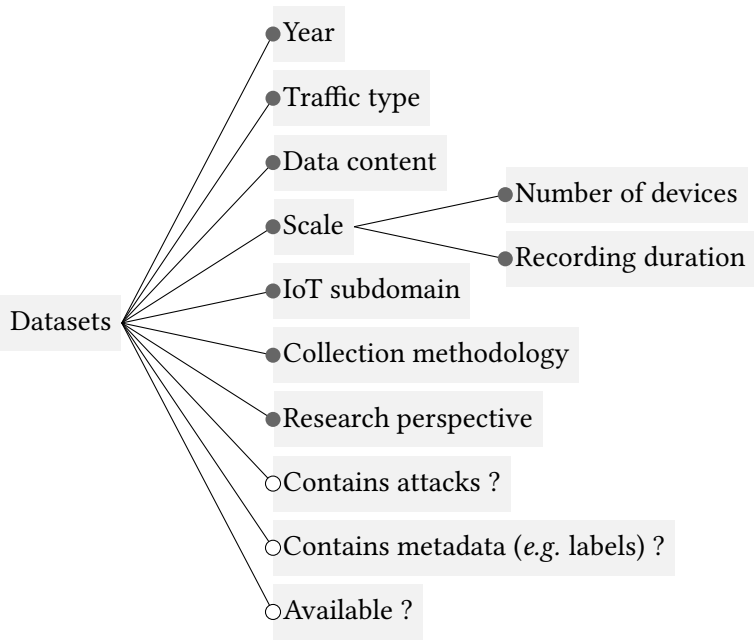


Figure 3.4: Feature diagram of dataset characteristics: first level

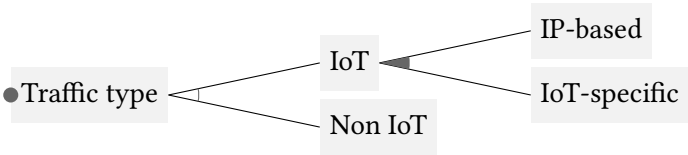


Figure 3.5: Feature diagram of dataset characteristics: Traffic type

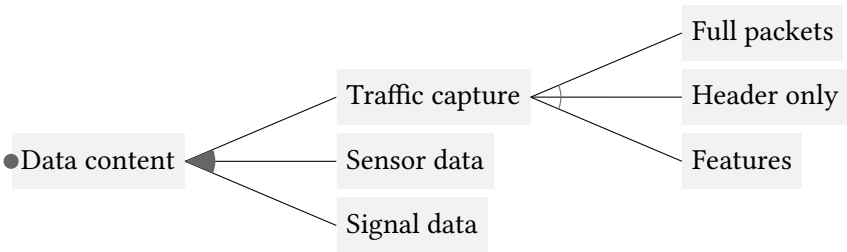


Figure 3.6: Feature diagram of dataset characteristics: Data content

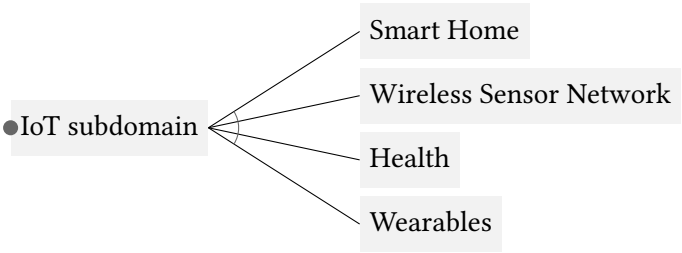


Figure 3.7: Feature diagram of dataset characteristics: IoT subdomain

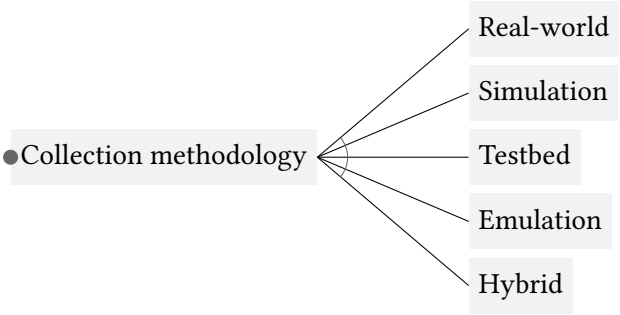


Figure 3.8: Feature diagram of dataset characteristics: Collection methodology

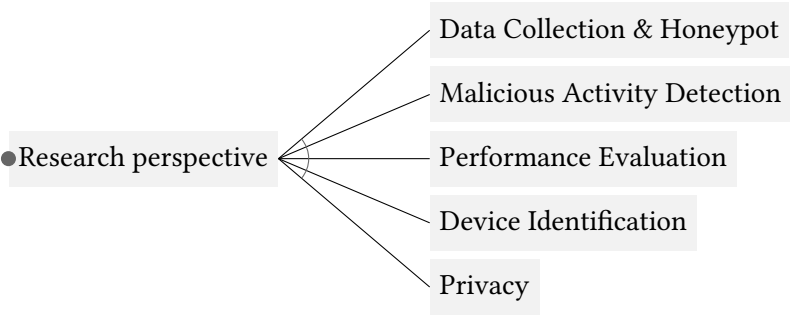


Figure 3.9: Feature diagram of dataset characteristics: Research perspective

using IoT devices, the traffic type can be further split based on whether the recorded network protocols include classical IP-based or IoT-specific ones. The IoT-specific protocols have been described in Section 2.1.3.

The *Data content* refers to the nature of the data contained in the dataset, which can reside at different abstraction layers of the communication. This can range from data about the physical electromagnetic signal, to high-level environmental data measured by IoT sensors, or covering traffic data from various layers of the OSI network layer model [36]. In the latter case, the data can be of various precision levels: it can contain the full network packets (including the full payload) in the form of packet captures, only the header of the exchanged packets, or not contain information about individual packets but traffic descriptive features which have been computed on the packets and flows.

Finally, the *presence of attacks* (labeled as *Contains attacks ?* on Figure 3.4), is a *boolean* value indicating if the covered dataset contains attack patterns or only benign traffic.

3.4.2 Scale

The scale of a dataset can be described more explicitly by the number and types of monitored devices and the duration of the data collection.

3.4.3 Collection methodology

This feature embodies the methodology the researchers used to collect their dataset. Multiple methodologies are possible, with different accuracies in representing real-world traffic:

- **Real-world network.** The most accurate with respect to real networks, but also the most difficult one, is to directly collect measurements from an existing network. This methodology might require a collaboration with an external organization, *e.g.* an ISP, to obtain access to an existing real-world network.
- **Simulation.** The researchers used a network simulator, and collected packets from the simulation. This method has the advantages of being easier and cheaper to setup, and also to reproduce. However, the traffic being simulated, it less accurately portrays real-world traffic. Existing network simulators for this purpose include Riverbed Modeler [177] (previously known as OPNET), NS-2 [178] and NS-3 [179], or Cooja [180] for WSNs.
- **Testbed with physical devices.** The researchers constructed an experimental network containing real devices (hosts, servers, routers, swit-

ches, etc.), and the dataset was collected by monitoring this network. The advantage of this method is that real devices are being monitored in a controlled environment. It therefore captures well the behavior of the used network protocols. Its drawback is that the deployment of physical devices is expensive and requires a physical place where the testbed will be set up.

- **Emulation.** Instead of using real, physical devices in their testbed, which can be expensive, researchers could emulate the necessary devices with the help of more common hardware. For example, they can instrument general-purpose devices or virtual machines to emulate IoT devices, virtual switches/routers by using SDN, or even generate relevant network traffic by using specific traffic generation tools.
- **Hybrid.** To combine advantages of different methodologies, researchers can also deploy a hybrid testbed, composed of real, emulated and simulated devices.

3.4.4 Year

As networking and security are two quickly evolving topics, a dataset recorded ten years ago probably contains traffic patterns that will not occur as much today, or attacks which are not possible to perpetrate anymore due to security updates. As such, the age of the dataset is an important metric to assess the representativeness of a dataset compared to real-world traffic.

3.4.5 Presence of metadata

To provide more insight into the network traffic contained in a dataset, some authors process the raw network traces to extract traffic-descriptive *features*, e.g. timestamps and nature of benign or attack traffic, number of bytes or packets exchanged, addresses and ports used, etc. This additional information can help further analysis of the data.

Nowadays, a majority of the collected datasets are used to train and/or evaluate ML models for intrusion detection or traffic classification. For such use cases, the data must be *labeled*, i.e. the data points in the dataset must be marked as benign or malicious (for intrusion detection) or as belonging to a specific device type, protocol, or application (for traffic classification).

3.4.6 Availability and usage by others

Availability tells whether a dataset is (publicly) available to other researchers.

In addition, we have checked whether the dataset has been used by other researchers than the ones that have produced it. Although this is not an intrinsic property of the dataset by itself (and therefore not shown in Figure 3.4), it can indicate whether the dataset is of good quality or has interesting content, and can be used in other cases in a relevant way. However, note that several older datasets, *e.g.* KDD Cup 99 [170], have been used a lot in the past, not because they are of good quality or relevance, but because researchers did not have a wide corpus of datasets to choose from. Vice versa, recent datasets might not have been used by others, not because they are of bad quality, but because they are too recent and not yet well known. For this characteristic, we searched for papers that cite the paper where the dataset is presented. If one of those papers indicated that it used the dataset in question, there is a match. If there is no match, we indicate it by a question mark. It does not mean that nobody has ever used it, but that we could not detect any usage.

3.4.7 Related research perspective

The datasets discussed in this survey were created by their authors for different motives. Therefore, their usefulness to other researchers will depend on the targeted research area. Based on the content of the datasets and the authors' descriptions of their goals and intentions, we identified five main research perspectives that could benefit from the datasets.

- **Data Collection & Honeypot** includes datasets that have been built with the specific intent of creating a dataset to fuel further research, or to fill a gap in the state-of-the-art. Honeypots are intentionally vulnerable systems, deployed specifically to collect data from attackers that will potentially attack the system [181]. As such, they also fit in this category.
- **Malicious Activity Detection** refers to datasets built for the evaluation of security systems which detect or prevent malicious activity, such as Intrusion Detection Systems [182] or firewalls [183]. Those datasets might have been produced to directly evaluate the author's system, or for future usage by other researchers.
- **Performance Evaluation** indicates datasets which have been created along the evaluation of a system or protocol, but not necessarily related to security.
- **Device Identification** relates to captures of device activities which have been used or can be used for the development and evaluation of device identification algorithms (*e.g.* [184]).

- **Privacy** is associated to datasets resulting from privacy analysis of devices, *i.e.* analysis concerning the disclosure of sensitive user data during device operation [185].

It should be noted that, even if datasets have been created for a specific purpose, they might be used in other research domains if relevant.

3.5 Overview of datasets

In this section, we provide a condensed overview of the covered datasets, using tables and plots aggregating data from all datasets. Tables summarizing all datasets are included in Appendix A.1.

3.5.1 Summary of research perspectives

Table 3.2 classifies each dataset along the research perspective it was initially produced for. Nonetheless, any dataset might be used in the context of another research perspective, if deemed relevant. We identified five different perspectives:

- **Data Collection & Honeypot:** Pure data collection without a specific purpose. May contain both benign and attack traffic.
- **Malicious Activity Detection:** Any usage related to intrusion/anomaly detection, *e.g.* ML model training and testing, or IDS evaluation.
- **Performance Evaluation:** Evaluating the performance of different protocols, network architectures, etc.
- **Device Identification:** Categorizing or identifying different devices according to their behavior.
- **Privacy:** Protection of sensitive user data.

3.5.1.1 Protocol distribution

Figure 3.10 shows, per protocol, the number of occurrences in IoT-related datasets. Only protocols which occur more than once have been included. Datasets not directly related to IoT have not been included in this measurement.

We observe that the majority of datasets contain IP-based traffic. This is expected, as IP is the building block of numerous higher-layer protocols, including general-purpose ones such as TCP and HTTP(s), but also the IoT-oriented MQTT. However, the sum of occurrences of all other, non-IP protocols, is 31, a number close to, and even higher than the IP-based one.

Research	Datasets
Data Collection & Honeypot	[186], [187], [188], [189], [190], [191], [192], [193], [194], [195], [196], [197], [198], [199], [200], [201], [202], [203], [204], [205], [206], [207]
Malicious Activity Detection	[208], [209], [210], [211], [212], [213], [214], [215], [216], [217], [218], [219], [220], [221], [222], [223], [224], [225], [226], [227], [228], [229], [230], [231], [232], [233], [234], [235], [171], [236], [237], [238], [239], [240], [241], [242], [243], [244], [245]
Performance Evaluation	[246], [99], [247], [248]
Device Identification	[249], [250], [251], [252], [253], [254], [255]
Privacy	[256], [257]

Table 3.2: Datasets classified per research perspective

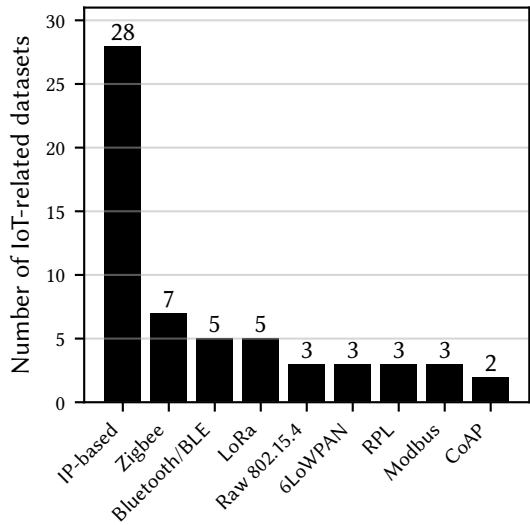


Figure 3.10: Distribution of protocols in IoT-related datasets

3.5.1.2 Number & category of devices

For many use cases, the types of devices and the number of devices or hosts appearing in a dataset are important characteristics. Obviously, researchers building an IDS for SH networks are interested in datasets containing traffic from such devices. On the other hand, datasets including many sensors are more likely suitable for large-scale experiments. Figure 3.11 shows the number of devices appearing in each dataset, sorted by device category.

We observe that most datasets contain data from networks with one to 50 devices. Note that some datasets contain more than one type of devices.

3.6 Challenges and Future Directions

In this second to last section, we highlight the challenges related to the datasets we covered, and propose further research directions. Namely, we point out the lack of datasets including IoT-specific protocols, the relative small scale of the presented datasets, and the challenges associated with data privacy and the generation of realistic traffic.

3.6.1 IoT-specific protocols and attack traffic

In Figure 3.10, our survey has highlighted the prominence of IP-based datasets and the lack of datasets involving IoT-specific protocols. We have found five datasets with Bluetooth traffic, seven with Zigbee, and five with LoRa. On the application layer, the CoAP protocol seems interesting in theory, but appears in only two datasets, and with a limited variety of attacks targeting it. This could indicate that the protocol is not widely used in practice, or has not drawn enough attention from the industry or researchers. To summarize, for many protocols at all networking layers, there is only little public data available that would allow researchers to see how those protocols are actually used in the wild and that would allow performing experiments that can be reproduced by others.

In addition, some of those datasets have limitations that might make them less useful for security researchers. Taking the example of LP-WANs, *i.e.* LoRa and Sigfox, most of the relevant datasets were created for device fingerprinting and localization [258, 259, 260, 261, 262]. As such, those datasets only contain physical layer data, as well as potentially the geolocation of the devices, but they do not contain network data such as the payload or port used for data transmission. Moreover, at the time of this study, no public dataset existed that contains attack traffic targeting such networks, and none has been produced yet as of the writing of this thesis.

We therefore hope that future researchers working in this area will consider generating and releasing datasets that contain such protocols. The in-

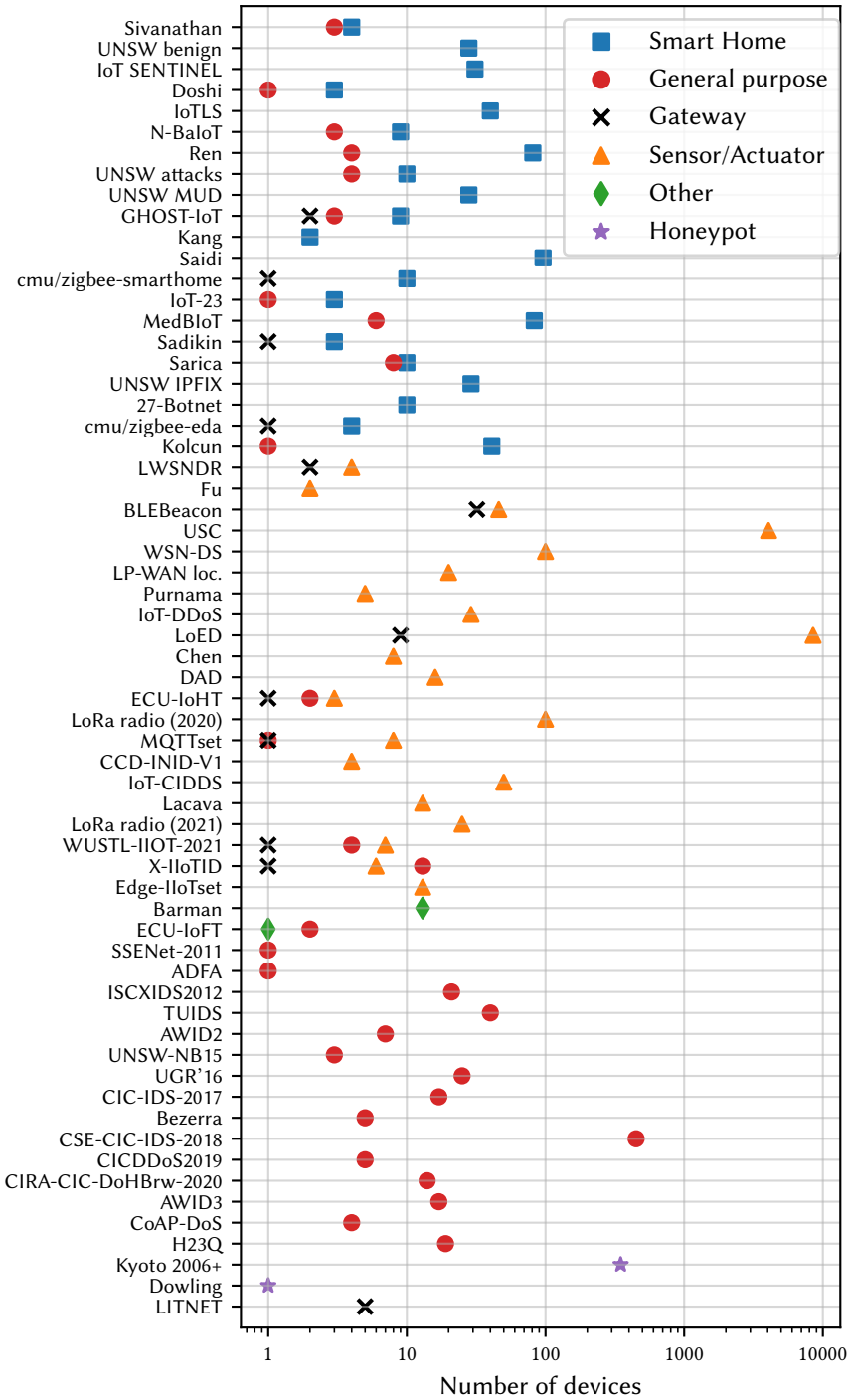


Figure 3.11: Number & category of devices used per dataset

centive should be given to the research community to produce more network datasets especially for IoT-specific protocols, such as LP-WANs and CoAP, and consider including attack traffic, preferably of diverse types. Publicly releasing such datasets can prove very helpful for research on the network security of IoT devices.

3.6.2 Data privacy and confidentiality

IoT devices have become popular in consumer and industrial applications, and given the nature and amount of data exchanged in IoT environments, data privacy and confidentiality have become important concerns.

Data privacy refers to the property that no information related to an individual should be disclosed without their explicit consent. An example of why this property must be enforced is in healthcare applications. IoT devices collect data such as the user's heartbeat rate, their blood pressure, and their physical activity. With such data, sensitive information about the user can be retrieved, *e.g.* the presence of a disease, and be used in a harmful way [263]. *Data confidentiality* ensures that unauthorized people cannot access the data. The difference between data privacy and confidentiality is that the former protects individuals while the latter protects the data [264].

Ensuring privacy and confidentiality is important in IoT environments, however, there exist challenges that hinder this. For example, a fundamental aspect of data privacy is that users understand what data is collected about them and where it is processed. This is rarely the case in current consumer-oriented IoT applications for various reasons. One of them is the simple fact that users do not always have the skills and knowledge to understand all the technical processes in IoT environments, and the manufacturers do not always make the effort to provide simple and clear explanations [265]. As far as data confidentiality is concerned, the main challenge is the limited resources of current IoT devices. In traditional networks, confidentiality is ensured by cryptography. However, implementing complex cryptographic mechanisms has been historically difficult on IoT devices due to their limited computing power, though this is slowly changing thanks to the increasing affordability of more powerful microcontrollers, and motivated by the traction around novel IoT protocols requiring stronger protection, namely Matter [78]. In addition, the heterogeneity of devices and communication protocols makes it difficult to design solutions that are compatible with any type of IoT environment.

The goal of the datasets presented in this survey is mostly to evaluate anomaly detection techniques, as shown in Table 3.2. Indeed, only two datasets were created with a privacy-related research purpose. Privacy is a relatively new topic compared to data confidentiality and computer security in general. Clearly, there is a lack of interesting datasets for research on privacy.

However, researchers are often not willing or allowed to publicly release their datasets because of privacy and confidentiality concerns. When releasing a dataset, complex measures must be taken to not violate privacy. If the dataset is empirical and therefore the result of the involvement of individuals, it is necessary to inform them which data are collected and what they are used for, and the necessary steps must be taken to get their explicit consent. Data confidentiality can be also problematic, especially in datasets that contain traffic payload, as necessary for the evaluation of IDSes using Deep Packet Inspection (DPI). If such a dataset is released, the authors must be careful about the information that can be retrieved in the dataset.

To avoid these challenges, two approaches exist: one is to rely on test beds or simulations without human involvement, for example by using traffic generators (see Section 3.6.4). The other is the usage of anonymization techniques. Cermak *et al.* [266] proposed a partial solution to this problem by introducing a methodology to be able to share datasets while keeping private information hidden. Their methodology proposes the usage of small portions of data, which they dub “annotated units”, which represent a subset of a full, real-world traffic trace. Those units can be anonymized more easily, and thus be shared with the community. For example, such a traffic unit can be the subset of the full network transmission that corresponds to one specific type of attack. Researchers willing to use those can then enrich the small units with potentially private self-collected traffic to form a dataset of a relevant size. In any case, when releasing a public dataset, a trade-off must be found between its usefulness and the level of anonymization applied to it [267, 268, 269].

3.6.3 Scale and complexity

As shown in Figure 3.11, most IoT datasets are relatively small in terms of number of monitored devices. This is understandable for applications such as Smart Homes that only involve small networks. However, it’s less understandable for other application areas where the scalability of IoT to networks of thousands of devices has been demonstrated by the industry. We therefore appeal to the research community to collect and publish more datasets from such large-scale applications.

Another open issue is the complexity of existing datasets. Most of them focus on a specific communication technology. However, modern IoT applications are often deployed in heterogeneous systems, even on a small scale. For the latter, a typical example is a Smart Home containing devices using various technologies, from short-range personal-area networks to long-range and cellular networks. In such a system, it would be wrong to see the individual networks as isolate entities since the devices are interdependent. Future

datasets have to reflect this complexity.

3.6.4 Realistic traffic generation

Recording a dataset from a testbed or operative network requires considerable effort, not only to configure and instrument the infrastructure but also to guarantee that the monitored data is representative and pertinent, *i.e.* it contains realistic network traffic caused by benign as well as malicious activities. To alleviate this task, one may draw support from *traffic generators*, which are tools able to synthesize benign and malicious traffic.

In the past years, a large number of tools for the generation of general IP traffic have been published, including the open-source D-ITG [270, 271], Harpoon [272, 273], fudp [274], and TRex [275]. However, those well-known tools are more inclined towards classical Internet traffic, and might not excel at simulating IoT traffic. Recently, researchers started developing traffic generators oriented towards IoT networks and protocols, six of which will be presented below.

IoT-Flock [276, 277] is an open-source framework able to generate CoAP and MQTT traffic. It can produce benign and malicious traffic. Use cases can be defined in XML files, or by using the graphical or command line interfaces. The authors evaluated their tool by simulating their IoT sensor testbed, as well as malicious traffic. Both types of traffic were discriminated very accurately by ML classifiers.

The hybrid IoT traffic generator by Hsu *et al.* [278] is based on an analysis of human-to-human activities. It can generate Machine Type Communication (MTC) traffic with geographic information attached for cell and network loading level assessment. To showcase its usability, the authors applied their generator on two related examples: mobile network performance assessment and improvement.

Hui *et al.* [279] used a condition mechanism to incorporate semantic and network structure knowledge and device category in IoT traffic generation with a generative adversarial network. The generator consists of three sub-generators to satisfy three of the requirements for IoT traffic generation mentioned in their paper. Their evaluation shows that their traffic generator comply with state-of-the-art data fidelity baselines.

Nguyen-An *et al.* [280] extended Scapy to generate IoT traffic based on the functionalities and characteristics of different devices in SH environments. All the generated packets are gathered into a single trace file, with support for different formats such as PCAP and CSV. For the evaluation, the authors simulated SH traffic, which resulted in a lower entropy in network parameters for their simulated traffic than for real traffic.

The LoRa traffic generator by Lavric *et al.* [281] generates packets that are

compatible with the LoRaWAN specifications. The generator is able to generate large volumes of traffic at high speed. The user has access to parameters such as spreading factor, packet payload, and coding rate. The traffic generator can be integrated into other applications. Thus, gateway manufacturers can use it to assess the real throughput of their products.

The LoRa cell traffic generator by Gucciardo *et al.* [282] is able to emulate the behavior of thousands of low-rate sensor nodes deployed in the same cell by using a single Software Defined Radio (SDR) platform. Unlike the traditional generators who focus on creating individual packet flows which emulate specific applications and protocols, this generator generates a combined radio signal given by the super-position of the signals transmitted by multiple sensors simultaneously active on the same channel.

A challenge that has to be addressed by future traffic generators is the fact that the different networking layers in IoT applications are highly intertwined. IoT application traffic over a LoRa network is not just a slower version of the same traffic over a Wi-Fi network. Resource constraints force developers of IoT applications to adapt the network communication to the limitations of the used devices and network technologies. Therefore, the generation of realistic IoT traffic has to consider the targeted environment in its entirety, from the application down to the type of devices.

In parallel to traffic generation tools, one can retrieve relevant traffic data by instrumenting honeypot networks, which are networks intentionally left vulnerable and open to the internet, with the specific intent of collecting potentially malicious traffic. Franco *et al.* [283], analogously to our manuscript, surveyed existing honeypot networks for (Industrial) IoT networks, which could be used to gather relevant traffic for IoT network security research.

3.7 Conclusion

Datasets are critical in many research areas and the existence of a sufficient number of high-quality datasets can not only be the base of new scientific insights but also ensure the accuracy and reliability of the results. In particular, *public* datasets allow comparing solutions developed by different teams and to publish results that can be reproduced and verified by others.

In this chapter, we have presented a comprehensive survey of available datasets suitable for the research on IoT network security. Among the 74 surveyed datasets, 52 contain IoT network traffic, *i.e.* around 70% of the total number. 38 out of those 52 also contain network attacks. We have also included several well-known datasets that do not contain IoT traffic but that have been used in previous research on IoT, sometimes due to the lack of IoT-specific datasets. We have classified the datasets based on ten characteristics, such as the observed network protocols, the level of detail of the dataset

records, as well as the date, duration, number of devices and methodology of the data collection, in order to help researchers who are looking for a dataset suitable to their particular needs. Finally, we have also briefly described the existing tools to generate IoT traffic.

Our conclusion is that there is still a scarcity or even lack of publicly available datasets covering IoT-specific protocols, such as LoRa or CoAP. Other emerging protocols, such as QUIC, which have initially been designed for other use cases than IoT might present characteristics relevant for IoT communication and should therefore attract research interest as well. In addition, many of the existing datasets do not contain attack traffic. Although benign traffic datasets are still useful for security researchers, it requires additional effort and careful planning to augment them with attack traffic. We therefore encourage future authors of new IoT datasets to also include various attack scenarios in their experiments.

This study was published in the journal *IEEE Surveys & Tutorials* [1]. The journal article includes, besides the content of this chapter and of Appendix A.1, a thorough description of each covered dataset.

Finally, we reflect on the usability of the surveyed datasets for the evaluation of our own research. Some of those have been collected in environments close to the one we are considering, *i.e.* Smart Home networks. However, while their data is similar to the one we are dealing with when building our solutions presented in the upcoming chapters, said solutions were designed to act directly on the traffic. Evaluating them passively on existing data is not enough; we must assess their actual impact on live traffic. Therefore, we decided to build our own sample network which simulates a real-world SH setting. This test network is described in the following Chapter 4.

Construction of a Realistic Smart Home Testbed

4

In the previous chapter (3), we surveyed existing datasets in the area of network security, mostly focussed on IoT environments. However, our contributions, highlighted in the next chapters, must be able to dynamically interact with the network environment; indeed, part of their operation relies on dynamically blocking network traffic. To evaluate our contributions, we must analyze the output traffic after the traffic blocking has been applied. As such, taking an existing dataset would not be sufficient.

Consequently, we built and deployed a SH test environment, which closely represents a real-world SH network. It comprises various off-the-shelf devices, and is enabled with multiple SH platforms. Besides, we did an effort to automate interactions normally performed by a user. This chapter will delve into the details of this testbed.

In practice, the testbed is deployed as a LAN. An access point is at a central position, and all devices communicating with IP are connected to it, either wired (via Ethernet) or wirelessly (via Wi-Fi).

4.1 Devices

The first step in building the testbed was choosing the devices that will comprise it. To ensure the testbed mimics a real-world setting as closely as possible, we followed a couple of criteria:

- Devices must be off-the-shelf and easy to acquire, such that they are similar to those a user would get.
- They must cover different categories and manufacturers, to broaden the scope of the research.

The devices we ended up acquiring can be categorized as end devices or hubs. The former are the ones which actually sense their environment or perform a physical action, while the latter acts as a controller, or intermediary between the user and said end devices. We further split the end devices into four categories:

- Power plugs
- Light bulbs
- Cameras
- Other sensors

4.1.1 End devices

End devices represent the devices which either gather data about the environment, *i.e.* sensors, or perform a physical action, *i.e.* actuators. Those are the end of the communication chain.

4.1.1.1 Power plugs

Power plugs are one of the simplest devices one can think of in the context of a Smart Home. They usually provide one user interaction, namely toggling it on or off. Sometimes, they additionally provide metrics about their energy consumption. In the testbed, we employ four different power plug models; they are listed below, along with their communication capabilities.

- **TP-Link HS110** [284]: IP over Wi-Fi only
- **Tapo P110** [285]: IP over Wi-Fi only
- **WOOX Smart Plug** [286]: IP over Wi-Fi only
- **SmartThings Outlet** [287]: Zigbee only

4.1.1.2 Light bulbs

Light bulbs are similar to power plugs, but tend to be a bit more complex. Whilst they can be toggled on or off, they usually also provide the feature to change their brightness or their color. We utilize three different connected light bulbs:

- **Tapo L530E** [288]: IP over Wi-Fi only
- **Alecto Smart Bulb** [289]: IP over Wi-Fi only
- **Philips Hue color lamp** [290]: Zigbee only

4.1.1.3 Cameras

The two previous device categories provided similar functionality. Connected cameras, on the other hand, exhibit completely different features. Their basic behavior is to allow their video feed to be streamed over the network, allowing the user to watch it remotely. Contrarily to the aforementioned one-off functionalities, this requires a more continuous, and therefore stable traffic. This inherent difference in design might be translated in observable disparities in their network behavior.

In our testbed, we use three different commercial cameras, all using only IP over Wi-Fi to communicate:

- **Tapo C200** [291]
- **Xiaomi camera** (Chuangmi IPC019) [292]
- **D-Link camera** (DCS-8000LH) [293]

4.1.1.4 Other sensors

To wrap up the end devices, we also employ three passive sensors.

The **Tuya Wi-Fi Passive Infrared (PIR) motion sensor**, as its name suggests, is a motion sensor, leveraging infrared light to detect motion. It communicates only via IP over Wi-Fi.

The two others are SmartThings-branded devices, and both communicate using the Zigbee protocol. The first is a **SmartThings multipurpose sensor** [294] which we used as a door sensor, *i.e.* a sensor which detects whether a door is open or closed. The second one is a **SmartThings presence sensor** [295]. It is a small object, akin to a key ring, which should be carried by users. When it is in the SH network range, the related apps mark the sensor as present.

4.1.2 Hubs

A *hub* is a device which, as opposed to end devices, does not perform concrete home automation actions (such as the aforementioned ones by end devices). They act more as controllers of the SH deployment, Namely, they are usually used for the following purposes:

- Acting as an intermediary between the user and a device, by issuing commands to the device on behalf of the user, *e.g.* through voice control.
- Configuring complex automations by orchestrating interactions involving different devices.

- Translating a communication protocol to another, such that end devices can communicate regardless of the protocol. For instance, Zigbee traffic is translated to IP packets (and vice-versa). In this case, hubs are often referred to as *gateways*.

A single hub can combine multiple of those functionalities.

The simplest hub that we use is the **Philips Hue Bridge** [296]. It only provides the third of the aforementioned features: as Philips Hue objects only talk Zigbee, the bridge serves as a gateway and translates the Zigbee traffic to IP, via its Ethernet interface. Unfortunately, it cannot be used for this purpose with devices from other manufacturers, which limits its utility.

Analogously as the previous hub, the **SmartThings Hub** [297] is used as a gateway for SmartThings-branded products, which only communicate using Zigbee, by translating their traffic to IP, also through Ethernet. Nonetheless, this hub provides more functionality than the Philips Hue bridge. Indeed, it enables the Smart Home with the SmartThings home automation platform, providing home automation capabilities. Other devices, including from other manufacturers, can be configured in this platform, improving the range of capabilities it provides.

The last employed hub is the **Amazon Echo Dot** [298]. Contrarily to the two previous ones, its purpose is not as a Zigbee gateway, and it only disposes of a Wi-Fi interface. However, it provides extended functionality as the controller of a home automation platform, namely Amazon Alexa. It also supports a wide range of devices from numerous manufacturers, and provides various automation capabilities by configuring cross-device interactions. The specific version we used in our testbed is the *Second Generation*.

4.2 Smartphone apps

By itself, a SH device does not provide much more than its old-school counterpart. To enable its IoT capabilities, it generally requires to be configured by a smartphone app. The nature of the latter can vary, but its broad purpose stays the same. It can for example be an official app provided by the device's vendor, in which case it is generally used for all devices of the same vendor. On the contrary, one can use a third-party app; an example of those is home automation apps, which can control and configure devices of various vendors. In both cases, apps are often able to set up more complex home automation tasks involving multiple devices interacting with each other. In the remaining of this manuscript, any smartphone app used for the aforementioned purposes will be called a *companion app*.

Here is the list of all companion apps considered in this research:

- SmartThings [299]: home automation app from Samsung, supporting various objects from a wide array of vendors.
- Alexa [300]: home automation app from Amazon, supporting various objects from a wide array of vendors.
- Kasa [301]: official app from TP-Link, for basic TP-Link devices.
- Tapo [302]: other official app from TP-Link, targeted towards their *Tapo* subcategory of devices, but supporting also basic TP-Link devices.
- Tuya [303]: home automation app from Tuya. The Tuya platform provides a centralized framework to manage fleets of IoT devices, along with a generic, all-in-one firmware providing all supported capabilities, that the vendor must choose to enable or not. This app supports any device running the Tuya firmware, regardless of the vendor. In our case, those devices are the WOOX smart plug, the Alecto Smart Bulb, and the PIR motion sensor.
- Philips Hue [304]: official app for the Philips Hue family of smart lighting.
- Hue Essentials [305]: third-party app for the Philips Hue family of smart lighting.
- Xiaomi Home [306]: official app for Xiaomi devices.
- mydlink [307]: official app for D-Link devices.

Table 4.1 gives, for each device used, the companion app(s) leveraged to interact with it at any point of the research project.

4.3 Complete network

By assembling the building blocks described in the previous sections, we form a realistic SH network, which mimics one a real-world user would deploy. A schematic version of this network is shown on Figure 4.1.

We connect the IP devices, either via Wi-Fi or Ethernet, to a router acting as a wireless AP, running OpenWrt [120], an open-source Linux distribution for network appliances. The AP is in turn wired to a router, also running OpenWrt. The latter serves as a gateway between the home LAN and the Internet, and performs Network Address Translation (NAT). It also takes the role of DHCP server for then LAN. We chose this network architecture, instead of connecting the devices directly to the network gateway, such that IP addresses stay identical when traversing from one AP interface to another;

	SmartThings	Alexa	Kasa	Tapo	Tuya	Philips Hue	Hue Essentials	Xiaomi Home	mydlink
Power plugs									
TP-Link	✓	✓	✓	✓					
Tapo	✓	✓		✓					
WOOX					✓				
SmartThings	✓								
Light bulbs									
Tapo	✓	✓		✓					
Alecto				✓					
Philips Hue	✓	✓				✓	✓		
Cameras									
Tapo	✓	✓		✓					
Xiaomi								✓	
D-Link									✓
Other sensors									
Motion					✓				
Door	✓	✓							
Presence	✓	✓							

Table 4.1: Companion apps used for each device

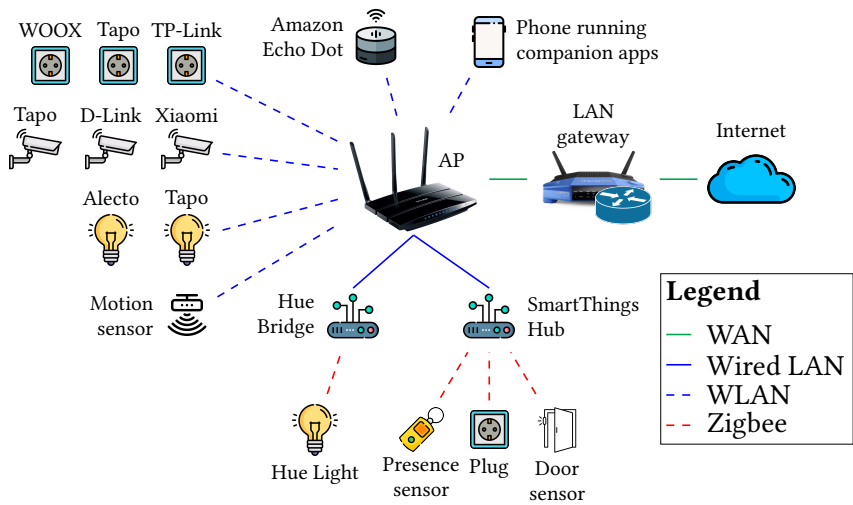


Figure 4.1: Complete experimental Smart Home network

this eases the deployment of our research-related software. Indeed, because of NAT, this would not be the case with the router acting as the LAN gateway.

The AP is a very modest dual-band AP, namely the TP-Link TL-WDR4900 (v1) [308], with an 800 MHz Freescale PowerPC Central Processing Unit (CPU) and 128 MBytes of RAM. The LAN gateway is a Linksys WRT1200AC [309], equipped with a Marvell Armada CPU running at 1333 MHz, and 512 MBytes of RAM.

Regarding the phone, we either used a Crosscall CORE-X4 [310], or a Google Pixel 9a [311]. The exact model will be specified for each experiment.

4.4 Interactions

What we call an *interaction* is any action performed with a given device, which has an effect on the state of said device. For an actuator, it consists in triggering one of its functions, *e.g.* toggling a power plug on or off. For a sensor, it simply consists in triggering its sensing capability, *e.g.* moving in front of a motion sensor. It can be triggered either by the user, or by another device in the network, if both are part of a home automation routine. In the latter case, the interaction is called a *cross-device interaction*.

4.4.1 Interactions per device

Each device provides a different set of possible user interactions, depending on its category:

- **Power plugs** provide only the *toggle* interaction, which toggles the plug on or off.
- **Cameras** provide only the *stream* interaction, which streams the camera's video feed on the companion app.
- **Light bulbs** provide up to three different interactions: toggling it on or off, setting the brightness, setting the color.
- **Other sensors** provide only one interaction, namely consulting their state.

Additionally, for all categories, we considered the *boot* interaction, which, as its name suggests, consists in booting the device, *i.e.* powering it on. Our testbed comprises devices which require a gateway to be able to communicate with IP networks, namely the SmartThings devices and the Philips Hue lamp. To perform the boot interaction with those, the end device and the gateway were booted conjointly.

4.4.2 Automation of interactions

The purpose of the construction of such an experimental testbed is that we can leverage it to perform our necessary experimental evaluations. Those require exerting interactions for one or multiple devices, which is usually done by manually interacting with the device's companion app, through the phone's touch screen. Doing so manually is very time-consuming and error-prone. Therefore, we made an effort to automate such interactions as much as possible. As those require interacting with the phone, we need a tool which is able to script user actions on a phone.

4.4.2.1 Tools: Android Debug Bridge & Maestro

Android Debug Bridge (ADB) [312] is a tool which can automate commands and user interactions with an Android smartphone. In practice, it starts a server on a computer, listening usually on TCP port 5037. This server, upon receiving a command from a client, will communicate with the ADB daemon running on the phone. Examples of useful commands are issuing a touch screen event at given screen coordinates, or taking a screenshot.

We leveraged ADB to automate interactions with the device, by scripting touch screen events at the correct coordinates on the companion app's screen. Those coordinates depend on the screen's dimensions and resolution, and therefore on the phone model. Manual configuration is required *a priori* to set those coordinates for all the necessary apps and interactions.

Maestro [313] is a tool, built on top of ADB, which eases testing the interface of Android applications. It allows, among others, simulating touch screen events on a given button by specifying its name, or even the text written on it, instead of the direct screen coordinates. Whilst this functionality would be helpful in automating the experiments for any phone, its execution is way slower than using plain ADB with the screen coordinates, and is therefore not practical. We settled on keeping using ADB for the sake of execution speed. Nevertheless, we leveraged the Maestro web interface for the *a priori* configuration: we used it to extract the relevant screen coordinates.

4.4.2.2 Verification of the interaction's outcome

Usually, an interaction's outcome is a change in the state of the device. However, due to the nature of our experiments, it might happen that the interaction is unsuccessful, *i.e.* fails to entail the expected change of device state. We designed a mechanism which detects if an interaction was successful, by checking the state of the device and/or companion app before and after the execution of the interaction. This mechanism obviously depends heavily on the device at hand.

Device	API	Screenshot
Plugs		
TP-Link HS110	python-kasa [316]	✗
Tapo P110	tapo [317]	✗
WOOX Smart Plug	✗	✓
SmartThings Outlet	pysmartthings [318]	✗
Lights		
Tapo L510E	tapo [317]	✗
Alecto Smart Bulb	✗	✓
Philips Hue	aiohue [319]	✗
Cameras		
Tapo C200	✗	✓
Xiaomi camera	✗	✓
D-Link camera	✗	✓

Table 4.2: Verification strategy per device

Our first strategy is to use the device’s Application Programming Interface (API), either official or third-party, if available. Such an API allows to accurately query the device’s state, *e.g.* the color of a lamp, or whether a plug is on or off. If no API is available, we resort to *screenshot-based verification*, similarly to related work [314]. This consist in taking a screenshot of the companion app before and after the interaction, and deciding if it was successful based on their comparison. We compare the screenshots using the Structural Similarity Index Measure (SSIM) [315], and set an empirically defined threshold to decide if the screenshots are equivalent. Said threshold varies depending on the companion app; typical values lie between 90% and 100%.

Table 4.2 gives the verification strategy used for each device with which we applied this strategy. In the case an API is used, the specific API is referenced.

4.5 Conclusion

Due to the nature of our upcoming contributions, which actively interfere with the SH traffic, reusing existing data would not be sufficient for a sound evaluation. For this reason, we built our own test network. For it to closely mimic a real-world SH installation, we equipped it with off-the-shelf devices that a regular user would consider acquiring. To provide variety, the devices span four categories: smart plugs, light bulbs, cameras, and sensors. Most communicate directly using Wi-Fi; the ones which do not use Zigbee, and

their associated hub translates it to IP-based traffic.

A couple strategies were implemented to maximize the testbed automation. First, the smartphone running the companion apps was instrumented with ADB: this enables running scripts which execute events on the phone, mainly touch screen presses. Then, we designed method to verify the state of all relevant SH devices, using the device-specific API if available, or screenshot-based validation otherwise.

In the following two chapters (Chapter 5 and Chapter 6), subsets of this complete testbed will be leveraged as a cornerstone of our contributions' evaluation.

A Profile-Based, Cross-Device Interaction-Enabled Smart Home Firewall

5

The Manufacturer Usage Description (MUD) standard [320], while being a great leap towards the standardization of behavior profiles for IoT devices, suffers from shortcomings which limit its usability in practical settings. In this chapter, we proposed a new format able to describe the intended communication patterns of IoT devices, inspired by MUD, and overcoming its limitations. One of the most compelling ways to leverage such profiles is as configuration for an allow-list firewall (see Section 2.2.2.3). To facilitate the usage of our format, we implemented a complete pipeline which generates functional firewall configurations taking as input a device profile complying with our format. We evaluated this pipeline in terms of validity of the generated configurations and performance of the firewall.

The work introduced in this chapter has been presented at the Networking Conference by the IFIP in 2024 [2].

5.1 Introduction

Cyberattacks against Smart Home IoT devices are a major concern for the privacy and security of their owners. Attackers could, for example, access IP cameras, sabotage smoke sensors, or manipulate smart locks. Unfortunately, the current state of the security of IoT networks in general and SH networks in particular is unsatisfactory. Due to resource constraints, it is often difficult to run state-of-the-art security solutions, such as host-based intrusion detection systems, on IoT devices [321]. In addition, the devices are for easy use by non-expert, so it often depends entirely on the manufacturer whether and how often security updates are applied and best practices are followed. Finally, their large number makes it possible for attackers to misuse them to perform powerful attacks against other Internet hosts [5]. Improving the security of Smart Homes has therefore become an urgency.

An assumption which is present in numerous research works [115, 152,

322] is that an IoT device has a network behavior governed by a limited set of network access patterns that are simpler than what can be observed for general purpose devices such as PCs. Consequently, one can express the device's expected network behavior in a compact form that we will call a *profile* in the following.

Leveraging this concept, the IETF defined the Manufacturer Usage Description (MUD) standard in RFC 8520 [320]. A MUD file is a description of the network connections allowed for a device. It specifies one or more traffic access control lists based on basic traffic features such as IP addresses, ports, connection direction, etc. The MUD file of a device can be used to configure local firewalls and switches such that the device can perform its normal functions, while other activities are blocked. Suggestions exist to extend MUD to support traffic statistics [323], such as packet rate or count, and some application-layer protocols [324].

A major weakness of MUD and its extensions is the fact that they do not sufficiently cover the reality of today's SH networks [325]. In the latter, devices can interact with each other directly or through one or more intermediate hosts. As a result, communication patterns emerge, consisting of temporally and logically interdependent connections. Such patterns cannot be described in MUD at all, and, consequently, malicious activities breaking those patterns cannot be blocked, which makes unauthorized device control and data exfiltration attacks possible [326, 327].

In this chapter, we present a system to efficiently enforce legitimate device interactions in SH networks. Our contributions are summarized as follows:

We propose a new language to express profiles that model legitimate network communication patterns of IoT devices. The language can express dependencies between traffic connections caused by device interactions.

We describe a firewall that leverages NFTables and NFQueue, lightweight firewall software available by default in Linux, to protect Smart Homes and a tool to automatically translate our device profiles into code for the firewall.

We evaluate our approach on different types of SH devices and show that our firewall correctly and efficiently forwards legitimate traffic while it blocks deviating traffic patterns with little overhead.

We publish the source code of our translation tool and firewall, the profiles that we wrote for the tested devices, and the traffic traces used in the evaluation, at <https://github.com/fdekeers/smart-home-firewall>.

It should be noted that it is not the aim of this chapter to prove that IoT device communications follow patterns that can be expressed in machine-readable descriptions. This has been discussed in the existing literature, to which we will refer in the relevant sections.

This chapter is organized as follows. Section 5.2 gives the motivation for our work and the considered threat model. We describe our system in Section

5.3. Section 5.4 presents the experimental setup for the evaluation of our approach, followed by Section 5.5 which presents our results. In Section 5.6, we discuss the strengths and limitations of our system, while comparing it to related work. The chapter concludes in Section 5.7.

5.2 Motivation and threat model

We introduce SH networks and the concept of device interactions, potential attacks, the limitations of MUD, and the threat model used in the rest of the chapter.

5.2.1 Smart Homes and device interactions

Smart Homes involve everyday household objects, from light bulbs to washing machines, enhanced with the capabilities of sensing (and, in some cases, acting), and communication. Such “smart” objects are connected to the LAN of the home, allowing them to communicate with each other, with other IT devices (e.g. laptops or smartphones), and with hosts on the Internet. The ones capable to speak the TCP/IP protocols can be connected through Wi-Fi or Ethernet. Some devices might only speak protocols specifically designed for local IoT communication, such as Zigbee [31], and need a gateway to communicate with IP devices.

Figure 5.1 shows an example SH network containing:

- a motion sensor, using Wi-Fi;
- a smart plug, using Wi-Fi;
- a smartphone (also using Wi-Fi) to control the SH devices with a vendor-specific companion app;
- a CPE supplied by the ISP that acts as a wireless AP and router.

Device interactions are a core principle of Smart Homes. To perform home automation tasks, devices need to seamlessly communicate with each other. In our example, the user might want the smart plug to turn on when the motion sensor detects movement. Figure 5.2 shows a version of the example SH network, including the network activities involved in this interaction:

- A1** The motion sensor indicates a movement detection to the cloud.
- A2** The cloud processes the message, and issues a command toward the smart plug.

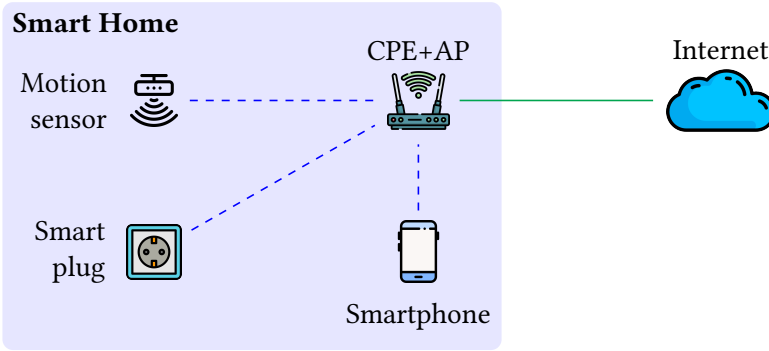


Figure 5.1: Example Smart Home network

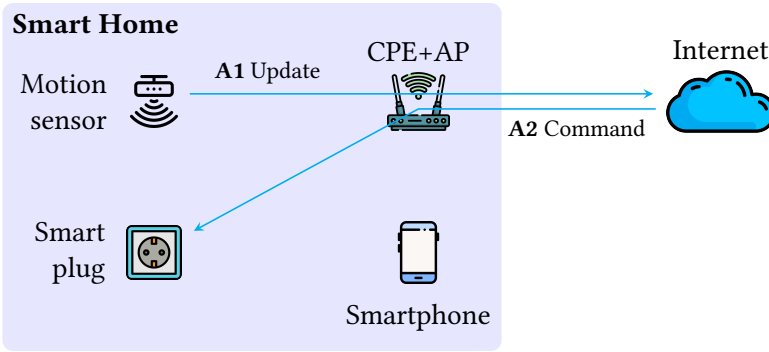


Figure 5.2: Example Smart Home network showcasing device interactions

In this interaction, the message that actually changed the plug’s state is the second one. However, this activity could not have occurred without the first one. Obviously, the network activities carry a certain semantic and therefore happen in a specific order between specific hosts. While a traditional firewall is limited to either allow or block each activity, any time and in any possible order, *an interaction-aware firewall, as proposed here*, can block communications that break the expected pattern.

5.2.2 Device profiling: the MUD format

This work focuses on devices which exhibit limited, specific, easily identifiable functions. For instance, a smart power plug must only toggle on or off, and optionally measure its own energy consumption. This simple behavior should also be translated in the communication patterns issued by the device. For the aforementioned functionality, the power plug should only communicate with the host which issued the toggling command, whether in the local

network or remotely. As such, the set of communication patterns the device is intended to issue is reduced, and can be expressed exhaustively, in the form of what researchers have dubbed its *profile*. One use case of such profiles is to use them as configuration files for network security systems using the *anomaly-based* approach, as described in Section 2.2.2.3: the traffic indicated in the profile should be accepted, while any other traffic should be rejected. Barrera *et al.* [21] performed early work in this direction, by proposing a simple profile format and a pipeline to convert the profiles into firewall configuration files for IPTables.

As this research direction was blooming, the IETF standardized a specification to express such profiles: RFC 8520, titled Manufacturer Usage Description (MUD). We take this standard as the base for our work. As such, we will delve a bit deeper in its format, and, most importantly, state its shortcomings that we intend to overcome. An example of a full MUD profile for the TP-Link HS110 smart plug is given in Appendix A.2.1.

Profile format MUD profiles are written as either Extensible Markup Language (XML) or JavaScript Object Notation (JSON) files. As such, they can be easily parsed by a program, while staying human-readable.

Profile content A MUD file comprises two sections. The first one is short; it contains metadata about the device at hand, *e.g.* the device identifier and version, a timestamp for the file, etc. The second one contains the core of the MUD file: the list of traffic patterns accepted for the device. It takes the form of two ACLs, formatted as specified in RFC 8519 [328], which describe the patterns issued by or toward the device, respectively. In turn, each ACL contains a list of packet descriptors, which specify the packets intended for a device. The MUD specification is designed for IP-based networks; only fields related to IP-based traffic are supported, including the following, per OSI layer:

- Layer 2: source and destination MAC addresses.
- Layer 3: IP (IPv4 or IPv6) source and destination hosts (IP addresses, networks, or domain names), and protocol; ICMP message type.
- Layer 4: TCP or UDP source and destination ports

Profile distribution The MUD specification does not only define the content of a device profile; it also describes an infrastructure to securely distribute said profile to a host which requests it, *e.g.* a router which would use the MUD file for firewall configuration. Basically, every device possessing a MUD file advertises a Uniform Resource Locator (URL), which can be queried to obtain the device's complete profile. For performance reasons, the host to

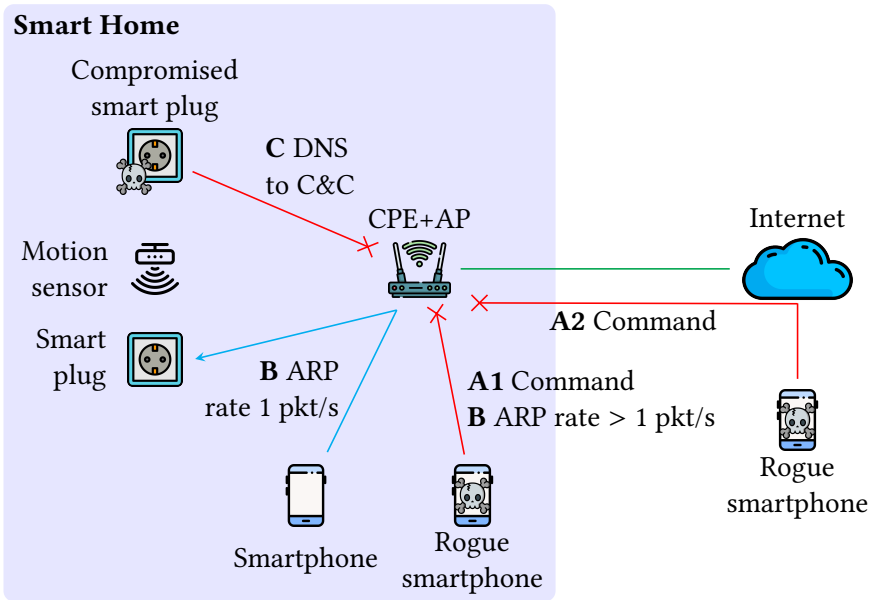


Figure 5.3: Example Smart Home network, including attacks

query should not be the device itself, but a more powerful server, able to process the queries (potentially as HTTPS) and distribute the profile. As MUD's name suggests, it is intended that the device manufacturer provides MUD files for its devices. In that case, the distributing server would be located in the manufacturer's cloud.

5.2.3 Limitations of MUD and potential attacks

The standardization of MUD is a big step towards a better security for SH networks, and even IoT as a whole. The rule of thumb is that traffic complying with a device's MUD profile is assumed to be legitimate, and vice-versa. Nevertheless, MUD suffers from a number of shortcomings, which allow attackers to issue malicious traffic which still complies with the profile. This section will detail three shortcomings, and, for each, present an example of an attack leveraging it. All three families of attacks are illustrated on Figure 5.3.

5.2.3.1 Sequential dependency between patterns

The biggest limitation of MUD is its inability to express a dependency between two subsequent traffic patterns. Each flow described in a MUD profile is considered as independent. This means that, if a flow is specified in the pro-

file, the flow is considered legitimate, regardless of other flows which occur before or after it. However, we introduced in Section 5.2.1 that, in a SH network, interactions between two or more devices are common. Those interactions involve several communication patterns which are not independent, but depend on each other. Even a single device might exhibit patterns involving a sequence of different traffic flows, each depending on the previous one: for example, a device first resolves the domain name of a remote server to its IP address via a DNS query, then communicates with this server using HTTPS. In both cases, a thorough security system might reject the second pattern if the former, which it depends on, has not been seen. This dependency between patterns is impossible to express with the current MUD format, and, to the best of our knowledge, no extension been proposed to include a similar concept.

We illustrate the need for such pattern dependencies in a profile with two variants of an attack involving sending a spurious command to a SH device. In a normal operation, the smart plug should only toggle whenever the motion sensor detects motion, according to the example depicted on Figure 5.2. However, with MUD, all communication between the smart plug and its controlling device, whether a smartphone in the LAN running the companion app, or the vendor's cloud, should be allowed, since it can occur in legitimate interactions. Consequently, an attacker could send spurious commands to the smart plug, as illustrated by the patterns denoted as **A1** and **A2** on Figure 5.3. They can do so either from the LAN (**A1**) or from the cloud (**A2**), if they manage to hack the cloud account of the SH user. The fact that the communication should be allowed only if it is preceded by a communication between the motion sensor and the cloud cannot be expressed in MUD.

On the contrary, our proposed profile language is able to describe legitimate communication patterns arising from device interactions, involving dependencies between sequential flows. Moreover, we design and implement a firewall system which can enforce them, therefore blocking any smart plug command that was not preceded by traffic from the motion sensor. Any unwanted traffic between the plug and its vendor's cloud, *e.g.* to exfiltrate privacy-sensitive data, would also be blocked by the same configuration.

5.2.3.2 Traffic statistics

The plain MUD format only supports describing the content of the packets. It cannot express meta-information about the traffic, *i.e.* traffic statistics such as the packet rate, the number of packets transmitted, the duration of the communication, etc. Such data might be important to prevent against attacks such as DDoS, which leverage packets complying with the expected format, but with a rate so high that the system cannot handle it. To overcome this

limitation, an extension to the MUD RFC was proposed, authored partly by the one of the initial MUD specification's authors, to add fields describing the allowed bandwidth tied to a communication pattern, more precisely the maximum rate of packet and/or bits [323].

Our second example attack highlights the need for descriptors in device profiles that allow to specify traffic rate limits. An attacker controlling a rogue device in the LAN can flood the network with unwanted packets (pattern **B** on Figure 5.3). In contrast to MUD, the profile language proposed by us allows specifying connection specific limits.

5.2.3.3 Protocols

While it intends to exhaustively express all communication patterns tied to the given device, MUD suffers from a lack of protocols and fields that it actually supports. First, it can only describe IP-based pattern, which means that non-IP protocols, which are common in IoT settings (*e.g.* Zigbee, LoRaWAN) are off the table. Then, within this already limiting category, it only supports protocols covering OSI layers 2, 3 and 4. Numerous protocols are impossible to describe with MUD, including all application-layer ones. It is true that most network traffic nowadays uses the HTTPS application-layer protocol, which is encrypted, and therefore could not be analyzed easily anyway. Nevertheless, many other unencrypted application-protocols are still in use, even more in IoT networks where the resources provided by the devices are limited, and the added cost for encryption might be overlooked. Widespread examples include DNS, HTTP and DHCP. An effort towards support for application-layer protocols was carried out by Matheu *et al.*, who extended MUD with the capability of matching application-layer protocols by their name, but they did not add support to match fields specific to those protocols (*e.g.* for DNS, the query type or queried domain name). Reddy *et al.* proposed an extension to MUD, which was standardized as RFC 9761 [329], to bring support for TLS features into MUD profiles.

We illustrate the usefulness of application-layer protocol descriptors in device profiles. We consider the scenario of an attacker who has infected an IoT device with botnet software. The device will try to contact a Command & Control (C&C) server on the Internet and send a DNS query to resolve its name. This is depicted as pattern **C** on Figure 5.3. With plain MUD, all DNS traffic must be allowed, since it is part of the normal behavior of devices. However, with our profile language, we can specify which domain names can be resolved.

5.2.4 Threat model

Our focus is on attacks that manifest themselves as changes in the communication characteristics and patterns of the devices. Such changes can be caused by the attack itself or be consequences of a successful attack. The considered attacks can be roughly divided into two categories:

- Attacks in which the attacker attempts to compromise, remotely control, or render a device unusable through the network. Examples include DoS attacks, scans, unauthorized remote access, and messages that were not initiated by the SH resident.
- Attacks in which the attacker misuses a compromised device to perform the above attacks against other devices or to exfiltrate data.

We assume that the attacker has the capability to attempt the above attacks from inside and outside the SH network. Obviously, our approach cannot protect against an attack that does not affect the network communication that is visible to the firewall. In the envisaged scenario where the firewall is deployed at a central point of the home network, this includes attacks using ad-hoc and long-range communication (e.g. 4G).

We assume that the attacker does not have the capability to (a) compromise the firewall software or hardware; (b) compromise the device profiles. The latter deserves further explanation. In the MUD standard [320], the idea is that the MUD file is provided by the manufacturer. It can be debated whether end users should trust the manufacturer when it comes to the protection of their Smart Homes. In fact, manufacturers have been caught collecting sensitive information from their customers' devices [330]. Therefore, the possibility that a manufacturer distributes compromised profiles cannot be excluded. We have designed our profile language such that the profiles are easy for experts to understand. We envisage a trusted platform from which profiles can be securely downloaded and where experienced users can review them, and report any security or privacy misuse, as a community effort to help less savvy users. Since this model has already been discussed for other use cases in the literature [330, 331], we consider such a platform to be out of scope.

5.3 System overview and description

We first give a high-level introduction to our security system for SH networks. After that, we describe its components in detail.

5.3.1 System overview

Our approach is based on extended traffic profiles that describe the allowed communication patterns, including device interactions, of SH devices. Our

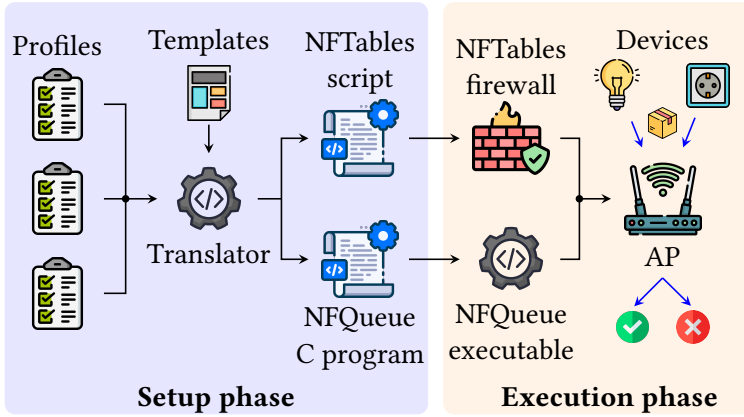


Figure 5.4: Overview of the system

goal is to leverage the information contained in the traffic profiles to identify and block unwanted or unexpected communications. We assume that the Smart Home to be protected is based on a (wireless) LAN infrastructure. An obvious place for a firewall is therefore the central home router or access point, where the communication between the devices and with the outside world can be monitored. If other (non-LAN) communication channels exist for the devices, they have to be monitored, too.

Figure 5.4 shows the components of our proposed system. It is intended for inspection of live traffic on consumer-grade home routers and APs. As such, for efficiency reasons, its workflow is divided in two phases: the *setup phase* and the *execution phase*.

The *setup phase* takes, as input, the profiles for the SH devices. Profiles are written in a human-readable language by the device manufacturer, the device owner, or other users. They describe the allowed communication activities at different protocol and abstraction levels. This comprises connection parameters (endpoints, protocols, etc.), traffic statistics (number of packets, byte rate, etc.), application layer information (e.g. URIs or domain names), and the relationships between connections (order, etc.).

The *execution phase* occurs when the firewall is deployed in the network, and inspects live traffic. Our firewall, running on a router in Figure 5.4, builds on NfTables [121], the Linux kernel firewall, to filter traffic from and to the SH devices. It installs NfTables rules that, where necessary, offload matching packets to a user-space program using the NfQueue infrastructure. That program is responsible for performing the complex matching and filter actions specified in the profiles.

During the *setup phase*, a translator (written by us) makes the interface between the device profiles and the NfTables and NfQueue-related files re-

```

device-info:
  name: my-device
  mac: 00:11:22:33:44:55
  ipv4: 192.168.1.100
patterns:
  dns-p:
    protocols:
      dns:
        qtype: A
        domain-name: my.server.com
      udp:
        dst-port: 53
      ipv4:
        src: self
        dst: gateway
    bidirectional: true
    stats:
      packet-count: 4
interactions:
  dns-https-server:
    dns-server: !include patterns.dns-p
    https-server:
      protocols:
        tcp:
          dst-port: 443
        ipv4:
          src: self
          dst: my.server.com
      bidirectional: true
      stats:
        rate: 50/second

```

Figure 5.5: Minimal device profile example.
 User defined labels are highlighted in green.
 Built-in keywords are written in black.

quired for the *execution phase*, by parsing the former to produce the latter.

5.3.2 Devices profiles

A device's profile is a model of its legitimate communication patterns. We use YAML as file format, instead of JSON or XML used by MUD. The reason for this choice is that the YAML syntax is more flexible, allowing, among others, the use of references to other fields in the same file. We extend this concept to allow referencing fields present in *other* files and also placeholders. This improves readability and reduces cluttering in the case of repeating patterns. To this end, our profile language supports a new include directive, which we have implemented thanks to PyYAML's support for custom YAML parsers in Python. It should be noted that the features supported by our profiles are a superset of MUD, *i.e.* any MUD file can be trivially translated into our profile format.

Figure 5.5 shows an excerpt from a profile. It consists of two main sections, namely the `device-info` section and the `interactions` section, that we describe in the following. The example also showcases the new `include` directive that allows to move parts of a declaration to another location in the profile and to refer to it by a qualified name. Concretely, the identifier `patterns.dns-p` refers to the user-defined field `dns-p` under the field

patterns. A larger device profile, again for the TP-Link HS110, is depicted in Appendix A.2.2.

The device-info section specifies metadata used to identify the device, that is its name, MAC address, and IP address (v4 or v6). In the remaining of the profile, the keyword `self` can be used to reference the device.

The interactions section defines a series of *interactions*. In the example on Figure 5.5, we only have one, which we (the profile author) have named `dns-https-server`. Each interaction defines a series of *policies*. In our example, the `dns-https-server` interaction defines two policies that we have named `dns-server` and `https-server`. We will first discuss them before explaining what interactions do.

The firewall only lets through packets matching a policy. The latter describes the packet features to match on. We allow all features supported by MUD (e.g. IP addresses, TCP/UDP ports, ARP message types, ICMP message types). In addition, we added support for more protocols, in particular for the application layer protocols (m)DNS, DHCP, HTTP, SSDP, CoAP, and IGMP, as these protocols are used by many SH devices.

To describe more complex network activities, multiple policies can be combined to an interaction (`dns-https-server` in the example). In an interaction, not all its policies are active at the same time. An interaction starts when its first policy sees a matching packet. That first policy becomes then the active policy of the interaction. The interaction will at some point deactivate its currently active policy and activate its next policy. How this is done depends on the type of the currently active policy:

- By default, a policy is a *one-off* policy. When it matches a packet, it is deactivated and the next policy is activated. If the property `bidirectional` is set, the policy requires a matching packet in each direction: it first matches on the fields as specified and then reverses the relevant fields (source and destination addresses and ports, DNS or ICMP message types, etc.) for the next packet.
- A *transient* policy has a maximum duration or maximum packet count specified in its `stats` section. The interaction moves to the next policy if the transient policy has reached the specified maximum duration or number of matching packets *or* when a packet is seen that matches the next policy. If the property `bidirectional` is set, the policy matches packets in both directions.
- A *periodic* policy contains a packet rate specification in the `stats` section. The policy stays active until a packet is seen that matches the next policy. Packets exceeding the specified rate are dropped.

Layer	Protocol	Field	Description
Net.	IPv4, IPv6	src, dst	Source & destination host (IP address or domain name)
	ICMP(v4)	type	Echo Request/Reply, etc.
	IGMP	version type group	IGMP version (v1, v2, v3) membership report, etc. Multicast group address
	ARP	type sha tha spa tpa	request or reply Sender MAC address Target MAC address Sender IP address Target IP address
Trans.	TCP, UDP	src-port, dst-port	Source & destination port (linked to their respective host)
App.	DHCP	type client-mac	DISCOVER, OFFER, etc. Client MAC address
	(m)DNS	qtype qname	A, CNAME, etc. Queried domain name
	HTTP	is_response method uri	Is it an HTTP response ? GET, PUT, etc. Requested URI
	CoAP	is_response code uri_path	Is it a CoAP response ? GET, PUT, etc. Requested URI
	SSDP	is_response method	Is it an SSDP response ? M-SEARCH, NOTIFY, etc.

Table 5.1: List of protocol fields, per layer, which can be leveraged by the firewall for its verdict

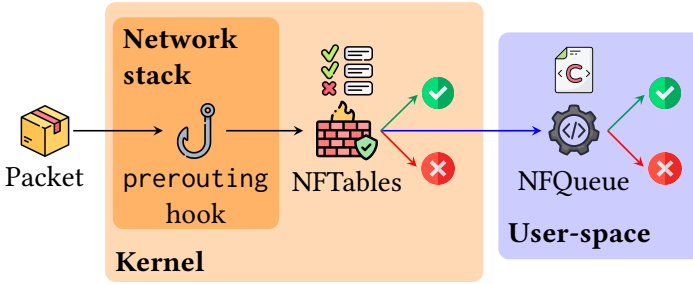


Figure 5.6: General NFTables/NFQueue firewall operation

5.3.3 The Firewall

The main added value of reduced profiles describing a device’s traffic patterns is its use as input for a security system. To leverage their simple profiles in this way, Barrera *et al.* [21] developed a pipeline to translate their profiles into IPTables rules. In the following, inspired by them, we describe our solution for a firewall that is able to enforce the traffic patterns described in our profile language.

As our profiles are much more expressive and describe interactions between devices, IPTables would be too limited to implement our concepts. Therefore, we chose NFTables, the next-generation Linux kernel firewall [121] and successor of IPTables. Since NFTables is shipped by default in many recent Linux distributions, our firewall can run on many platforms and does not require installing third-party tools. NFTables provides simple packet header matching based on rules covering the most prominent layer 2, 3 and 4 protocols, such as ARP, IPv4 and IPv6, TCP, UDP and ICMP, as well as matching capabilities based on rate and size. NFTables allows intercepting packets traversing the Linux network stack with *Netfilter kernel hooks* [332]. Our firewall registers to the prerouting hook, which is the first hook able to handle packets coming from different interfaces.

By itself, NFTables’ matching and filtering capabilities are quite limited. Its flexibility comes from one of its libraries, `libnetfilter_queue` (hereafter referred to as NFQueue), which allows NFTables rules to offload a matching packet to a so-called *queue*, implemented as a callback function in a user-space program [333]. The latter can, in turn, process the packet and issue the final verdict (*i.e.* accept or drop), as summarized in Figure 5.6.

We use NFTables’ limited packet header matching capabilities to execute the basic matching rules defined in the policies of our profiles, *e.g.* the matching by IP address or packet rate. We implement the advanced features of our firewall in external binaries that receive the packets from NFTables using NFQueue. Those advanced features are the following:

- The interaction mechanism described in Section 5.3.2.
- Matching on maximum duration and packet count, as needed for transient policies.
- Support for the protocols CoAP, DHCP, HTTP, IGMP, (m)DNS, and SSDP. All the protocol fields that the firewall is able to consider in order to issue its verdict are listed in Table 5.1.
- Support for domain names. Our firewall monitors DNS responses and caches the resolved names. This allows using domain names in policies instead of IP addresses, which is necessary for our profiles to stay up-to-date, as the IP addresses of the cloud servers contacted by an IoT device might regularly change.

When the firewall is running, there is one user-space process per device, containing the NFQueue callback code for that device. This process is further split into multiple threads, one per queue, that is, one per NFTables rule installed for that device. Note that a packet can match multiple rules, possibly even from several devices. For example, it could be affected by policies of the sending and of the receiving device. In that case, it will be forwarded to all corresponding NFQueue functions. The packet is admitted if it complies to any of the prompted policies.

5.3.4 Finite State Machine for interactions

The interaction-based matching and filtering mechanism described in Section 5.3.2 requires a stateful firewall. Our current prototype maintains one FSM per interaction to keep track of its progress. The number of FSM states depends on the number, types, and properties of the policies in that interaction.

We explain the functioning of the FSM with an example of an interaction with three policies *A*, *B*, and *C*. Policy *A* is one-off and bidirectional, *B* is periodic, and *C* is transient. The resulting FSM is shown in Figure 5.7.

State 0 is the initial state of the FSM. If a packet matching policy *A* is seen, we move to state 1. Since *A* is bidirectional, it now waits for the packet in the opposite direction. After that, it moves to state 2, which presents policy *B*. The latter is periodic, *i.e.* it will accept all matching packets and stay in state 2 until a packet matching policy *C* is seen, which causes the FSM to move to state 3. As a transient policy, *C* accepts all matching packets, but, unlike policy *B*, it will stop matching when the specified maximum duration or packet count is reached. In that case, the interaction restarts in state 0 or

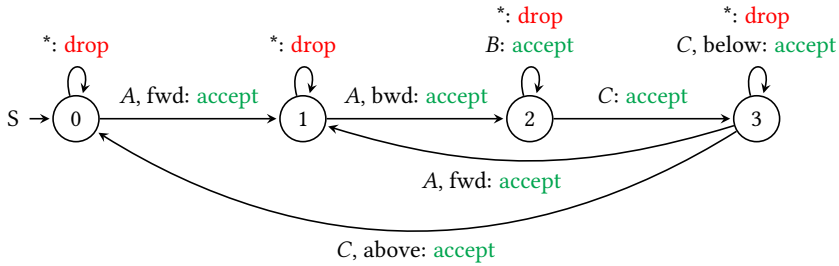


Figure 5.7: FSM for an interaction with a *one-off* policy *A*, a *periodic* policy *B* and a *transient* policy *C*.

“fwd” (resp. “bwd”) specifies the direction of the packet, namely “forward” (resp. “backward”).

“above” (resp. “below”) indicates a packet which should occur before (resp. after) in a transient policy.

“*” stands for all non-matching traffic.

the FSM directly moves to state 1 if the last packet matched policy *A*. All non-matching packets are dropped. Note that the rate limitation of the periodic policy is directly enforced by NFTables rules and not by the FSM.

5.3.5 The Translator

The *translator* is a tool written by us in Python that translates the device profiles, during the *setup phase* (mentioned in Section 5.3.1), into NFTables configuration scripts and C code for the NFQueue part of the firewall, that is for the features of the firewall beyond the simple matching rules of NFTables. To simplify the code generation, we have prepared C code templates that are completed with the information extracted from the profiles using the Jinja templating engine [334]. The C code is compiled to binary files that will be executed, upon deployment, on the device hosting the firewall (e.g. the network’s wireless access point).

5.4 Experimental setup

This section describes the experimental setup that we use for the evaluation of our approach. It is a subset of the complete testbed described in Chapter 4.

5.4.1 Devices

The employed devices are listed in Table 5.2. A schematic version of the testbed network is depicted on Figure 5.8.

Device	Category	Protocol
TP-Link plug	End device	Wi-Fi
Xiaomi camera	End device	Wi-Fi
D-Link camera	End device	Wi-Fi
Tuya motion sensor	End device	Wi-Fi
Philips Hue light	End device	Zigbee
SmartThings door sensor	End device	Zigbee
SmartThings plug	End device	Zigbee
SmartThings presence sensor	End device	Zigbee
Philips Hue bridge	Gateway	Ethernet, Zigbee
SmartThings V3 hub	Hub	Ethernet, Zigbee
Amazon Echo Dot (2nd gen)	Hub	Wi-Fi

Table 5.2: Testbed devices

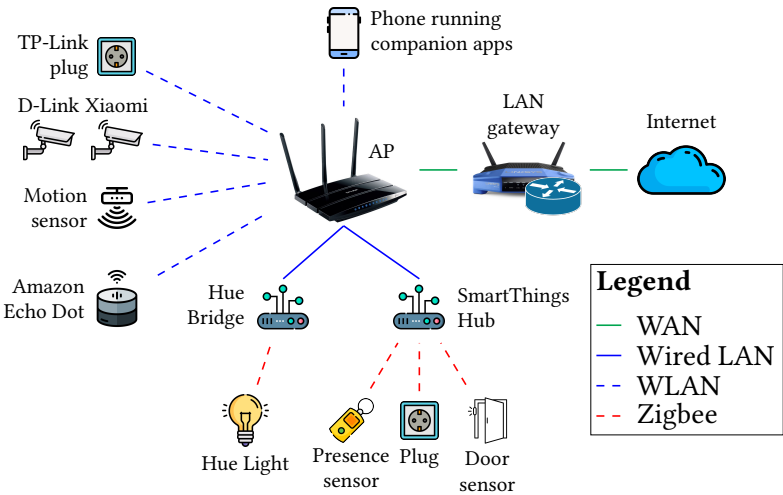


Figure 5.8: Subset experimental Smart Home network

5.4.2 Traffic capture

We capture the traffic from and to the devices. We use the resulting datasets for the creation of the device profiles.

To collect network traffic, we execute `tcpdump` on the four network interfaces of the LAN AP, namely the 2.4 and 5.0 GHz Wireless Local Area Network (WLAN), the wired local LAN, and the WAN interfaces. For the recorded traffic to be as exhaustive as possible, we physically interact with the devices. We also communicate with all devices through their respective vendor-provided companion app running on a mobile phone, first connected to the same LAN as the device, then to an external network. Device-specific commands are issued, *e.g.* switching the smart plug on and off, or streaming from the camera. For the devices compatible with the Amazon or SmartThings hubs, similar interactions are also done through the respective hubs' apps and through Amazon Alexa's voice control. In this way, we make sure to cover various types of device behavior, such as derived in [335]. Finally, we also leave the devices idle to capture background traffic and device activities not linked to user activities, such as checks for software updates.

5.4.3 Profile creation

We manually build device profiles from the traffic captures. Since the Philips Hue lamp and the three SmartThings end devices are not IP devices, their behavior is described in the profile of the gateway they are connected to, that is the Philips Hue Bridge and the SmartThings Hub, respectively. The Amazon Echo Dot controls the other devices via Wi-Fi. The legitimate traffic between the Echo and the Wi-Fi devices is rich and confined to the home. We thus do not create a profile for it. However, the Echo Dot *does* appear in the profiles of the other devices as part of their interactions with it. Figure 5.9 show the size of the manually created profiles for each device, in terms of number of policies and interactions contained in each profile.

The profiles are then automatically translated to the corresponding NFTables script and NFQueue C code files by our translation tool (see Section 5.3.5). Figure 5.10 shows, for each device, the size of the generated NFTables firewall, in terms of number of NFTables (NFT) rules and NFQueue queues (NFQueues). Figure 5.11 shows the size of the generated NFQueue source code, in terms of Lines of Code (LoC). The most “complex” devices are the Philips Hue bridge and the attached lamp. Unlike a smart plug, the lamp allows different types of interactions (*e.g.* adjusting the brightness, the color, etc.). Moreover, the bridge has more connectivity options (mDNS, SSDP) than simpler devices and can directly communicate with devices such as the SmartThings hub without an Internet connection. This results in a greater number of NFTables rules for the different protocols and in a larger NFQueue program to process the

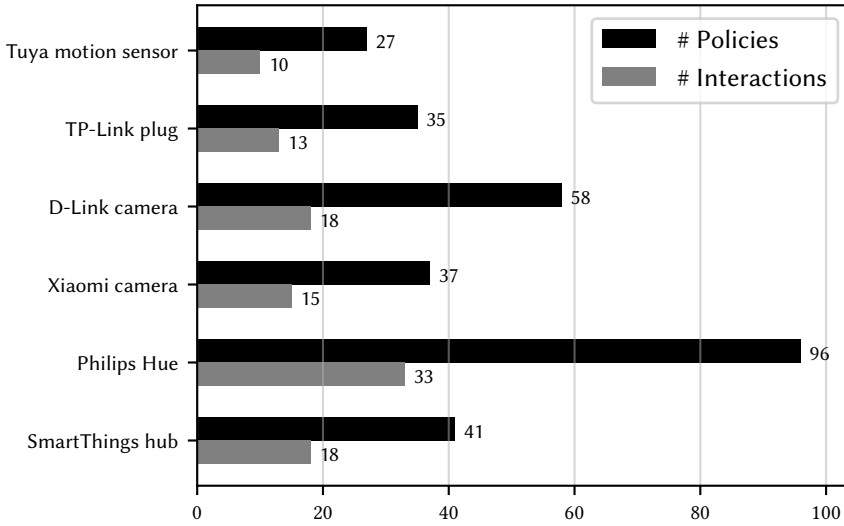


Figure 5.9: Profile size: Number of policies and interactions, per device

interactions.

5.5 Evaluation

We validate the performance of our firewall in attack-free and attack scenarios.

5.5.1 Fuzzing-based validation

We first verify that our system accepts traffic which conforms to the configured profiles, whilst dropping any traffic which does not conform. We developed a fuzzer which takes a packet capture as input and modifies protocol fields of randomly selected packets. The modification is always applied to the highest supported protocol layer (e.g. the DNS layer for a DNS packet) in order to stress the NFTables rules as well as the NFQueue code that handles the highest protocol layers.

Our testing methodology implements the following steps:

- **Base PCAP creation:** We take the device traffic collected as described in Section 5.4.2. The firewall should not block any of those packets.
- **Fuzzing PCAP files:** We run our fuzzer on each PCAP file five times to produce mutated versions.

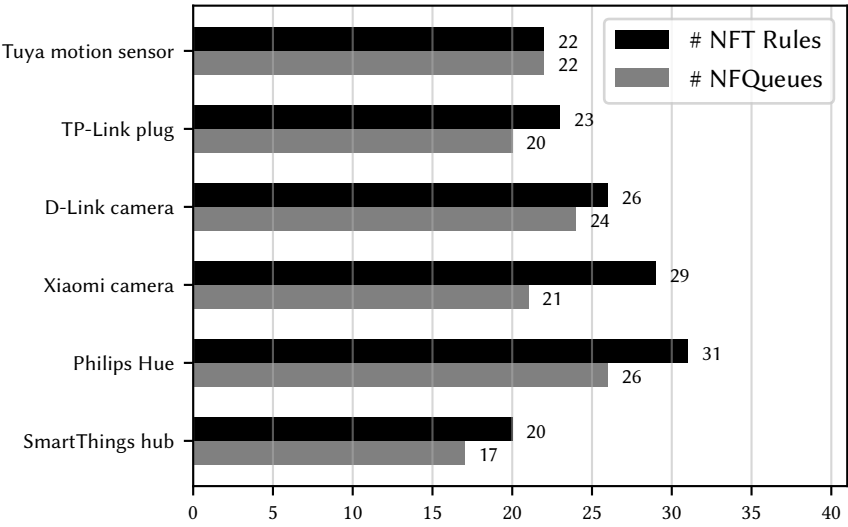


Figure 5.10: Firewall size: Number of NFTables rules and NFQueues, per device

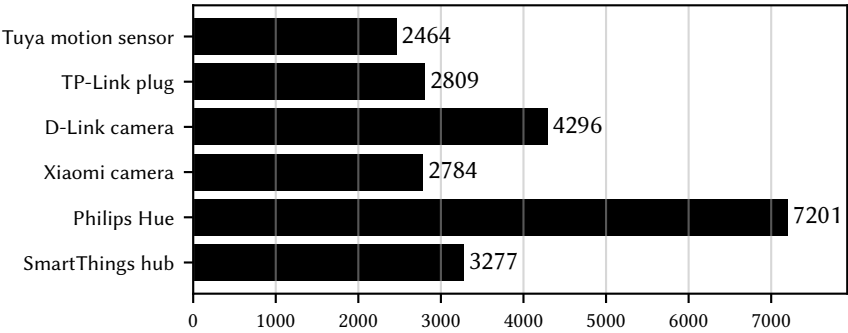


Figure 5.11: Lines of Code of NFQueue source code, per device

- **Labeling packets:** If the packet was modified and does not conform to the device profile anymore, its expected verdict is DROP. If the packet was *not* edited, we must check if a previous packet in the same interaction was modified. If yes, and if this should prevent the current packet from being accepted because it corrupts the interaction, the expected verdict is DROP, otherwise ACCEPT.
- **Replaying PCAP files:** We replay the mutated PCAPs through the firewall, log its verdict for each packet, and compare it with its label.

We replayed a total of 10,894 packets. Among them, 4,348 were supposed to be accepted, and 6,546 to be dropped. Our firewall issued the correct verdict for all packets.

5.5.2 Validation on attack-free scenarios

Our system must be transparent to the user of the SH network if no attack occurs. Traffic should flow without noticeable additional delay. To validate the correct behavior of our firewall in a realistic setting, we deploy it in the testbed depicted in Figure 5.8, consisting of the devices shown in Table 5.2. As explained in Section 5.4.3, the behavior of the Zigbee devices (the Philips Hue lamp, and the SmartThings plug, door sensor, and presence sensor) is embedded in the profile of their related gateway (Philips Hue Bridge or SmartThings Hub), whereas the Amazon Echo Dot is only used to control the Wi-Fi devices, and does not have a dedicated profile.

We measure the latency incurred by the firewall's processing on the AP. In practice, we compute it by recording the traffic on its four network interfaces, then taking the difference between the packet timestamps on the egress and ingress interfaces. Those measurements are performed in four different scenarios:

1. no-firewall: without any firewall.
2. base-firewall: with the default built-in OpenWrt firewall rules active on the AP. These rules accept all packets, but introduce some rate limiting, e.g. for incoming TCP SYN packets.
3. my-firewall: with our firewall. We only use one device at a time and activate only the profile for that device. We repeat this for all devices. The goal is to measure the influence of a device's profile on the firewall performance. Our firewall processes 2,105 packets (1.7 MBytes) for the D-Link camera (during 10 minutes, of which the camera was streaming during 20 seconds), 3,862 packets (0.8 MBytes) for the Philips Hue, 1,071 packets (0.2 MBytes) for the SmartThings hub, 682 packets (0.2 MBytes)

outliers reaching a maximum of over 3 ms. As the network latency typically lies between 2 and 5 ms for communications of devices in the same LAN, we deem the latency incurred by the AP negligible in most cases, and at worst on par with the former. We conclude that the latency caused by the firewall will virtually never be noticed by the user when interacting with the devices.

In the *my-firewall* scenario, the latency increases as our firewall augments the AP's processing. Moreover, in the *all-devices* scenario where the firewall is active for all devices at the same time, the firewall latency increases further. This is expected, as the system must process more packets and rules. Nevertheless, the latency stays acceptable, which indicates that our system is suitable for most current SH deployments, where the number of devices hardly exceeds ten [100].

We further investigate the origin of the latency induced by the firewall by splitting the processed packets in four categories, ranked in increasing intensity of firewall processing:

- A: packets which were matched by plain NFTables, without queueing to NFQueue;
- B: packets which were queued to NFQueue and required domain name lookup in a DNS map;
- C: packets which were queued to NFQueue and required string comparison;
- D: packets which were queued to NFQueue and underwent string comparison *and* domain name lookup in a DNS map.

We then count the number of packets corresponding to each category, and plot the latency for each category in Figure 5.13. We observe the joint effect of the sheer packet number and the category on the latency. Indeed, category **A** is the less processing-intensive one, but the fact that it has a lot of corresponding packets increases its total latency on par with the way more complex category **D**. Besides, the most time-consuming category is **B**. While it is more complex than **A**, triggering the user-space NFQueue program instead of only matching packets with NFTables, it is less complex than **C** and **D**; the most important parameter regarding its higher global latency is therefore the number of packets pertaining to this category, which is substantially larger than those for **C** and **D**. Finally, **D**-typed packets require more processing than **B** and **C** packets, but are less numerous, and thus less delayed on average.

5.5.3 Validation on attack scenarios

As argued in Section 5.2.4, our approach is, by design, limited to attacks that have an impact on the network activities. Such attacks include reflection at-

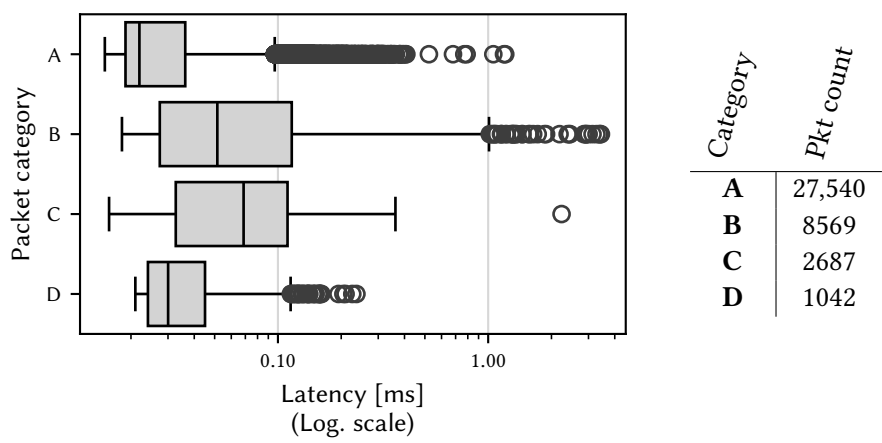


Figure 5.13: Firewall latency per category of processing intensity

tacks and other types of brute-force DoS attacks, network scans, data exfiltration, attempts to remotely control a device from an unknown host, send unexpected messages, etc. We do not show evaluation results for all of these attacks since many of them are trivially blocked because they use addresses, ports, or protocols that are not defined in the profiles.

In the following, we revisit the three attacks described in Section 5.2.3, as they concern key properties of our system, that is device interactions, traffic statistics, and support for higher-layer protocols. We run the attacks in the presence of our firewall and record the accepted and dropped packets. We also do latency measurements.

We implement the network shown in Figure 5.1 with a Tuya motion sensor, a TP-Link smart plug, and a smartphone. As explained before, the expected behavior is: The sensor, upon motion detection, transmits a message to the vendor’s cloud, indicating the detection. The cloud processes the message, and instructs the smart plug to change its state. We run the firewall with the corresponding profiles. Among others, this means that the firewall is aware of the above device interaction, of the expected traffic rates, and of the domain names of the servers (NTP servers, cloud servers, etc.) contacted by the devices.

We test the following attacks:

- A1: The attacker issues commands to the plug from a host located in the local network.
- A2: The attacker issues commands to the plug from a host outside the local network.

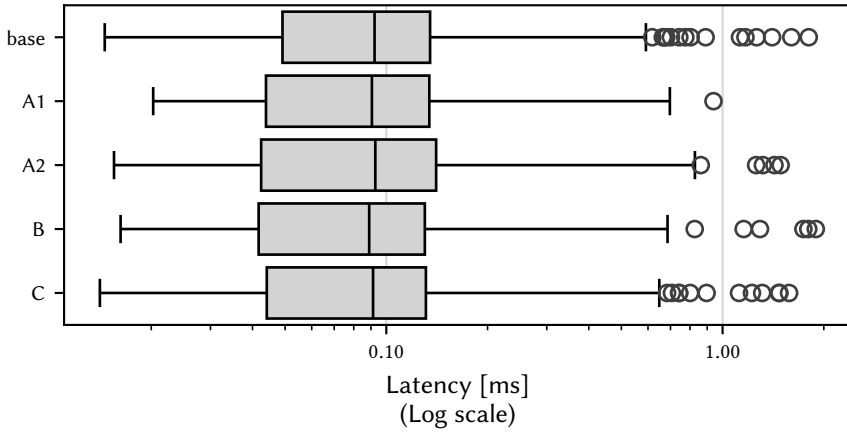


Figure 5.14: Latency induced by the AP hosting the firewall, per attack scenario

B: The attacker sends two ARP packets per second from the smartphone. The profile specifies a maximum rate of one ARP packet per second.

C: A device tries to resolve domain names of servers not mentioned in the profiles. This mimics a scenario where an attacker has compromised the device and lets it communicate with a malicious server.

The firewall successfully blocks all attack packets, without affecting the normal function of the devices.

Finally, we deploy again the full testbed network as illustrated on Figure 5.8, and we measure the latency incurred by our firewall, in a scenario similar to the all-devices one presented in Section 5.5.2, when the aforementioned attacks are ongoing. Figure 5.14 shows a box plot of the latency for legitimate packets, in the base case without attacks, as well as in the four attack scenarios. In all attack scenarios, the latency values seem comparable to the base, attack-free case. In order to verify their equivalence, we perform a *Kruskal-Wallis* statistical test over the five groups.

The *Kruskal-Wallis* test [336, 337] is a non-parametric test which tests whether two or more samples originate from the same distribution. Since it is non-parametric, it does not require normality of the samples' distribution. In our case, as observed on Figure 5.14, the distribution of the latency values is evidently not normal. The null hypothesis states that all sample groups are equivalent (*i.e.* originate from the same distribution), whereas the alternative hypothesis states that at least one group is different from the others. The null hypothesis is considered for rejection if the p-value is below a given thresh-

old (typically 0.05) [338, 339]. The *Kruskal-Wallis* test does not state which group(s) is/are different, nor the relation between groups.

The result of the *Kruskal-Wallis* test over our five samples is a rejection of the null hypothesis with a very low p-value of 1.17e-8, meaning the five groups are actually not equivalent, and at least one is significantly different from the others. However, when the sample size n is very large, which is the case here ($n = 38,255$), even the smallest, practically insignificant sample divergences might result in a statistically significant difference, the null hypothesis being systematically rejected [340]. To assess the differences between groups in a more practical way, we compute the ϵ^2 effect size [341], which can be interpreted as a coefficient of variation of the dependent variable (here, the latency) with respect to the independent variable (here, the attack scenario) [342]. This metric is defined as

$$\epsilon^2 := H \cdot \frac{n+1}{n^2-1}$$

where H is the statistic value, here resulting from the *Kruskal-Wallis* test, and n is the total sample size over all groups.

This results in $\epsilon^2 = 1.12\text{e-}3$; this value is considered very small, even negligible [343, 344]. While the statistical test concludes in a significant difference among the groups, the tiny ϵ^2 effect size tells us this result is mainly due to the large sample size, and that their practical difference is negligible. We conclude that the latency incurred by the firewall is comparable, whether in the attack-free or in the four attack scenarios. Therefore, network attacks have no meaningful effects on the firewall's operation: rejecting a substantial quantity of attack packets does not hinder its correct operation with legitimate packets.

5.5.4 CPU and Memory usage under stress

We measure the CPU and memory usage of our system to study its robustness and processing capabilities in case of higher network load. To this end, we deploy the TP-Link smart plug, along with our system configured with the profile of this device. We evaluate four scenarios with increasing computational resources requirements:

- **Normal:** Normal operation, *i.e.* identical to the latency measurements presented in Section 5.5.2.
- **State:** Under a steady flow of packets rejected by NFQueue because of the FSM state. The firewall must only check the FSM state to issue the reject verdict.

Scenario	Normal	State	String	Lookup
CPU [%]	<0.1	2.5	3.0	3.2
Memory [%]	5	5	5	5

Table 5.3: Mean of performance metrics

- **String:** Under a steady flow of DNS requests with an unexpected domain name. The firewall must perform name comparisons to issue the reject verdict.
- **Lookup:** Under a steady flow of packets directed towards an IP address not present in the DNS table. The firewall must perform a lookup in the DNS cache to issue the reject verdict.

For the last three scenarios, the packet rate is set to 1,000 packets/s.

Table 5.3 summarizes the measured values. We measure mean CPU usages of 0.1% (Normal), 2.5% (State), 3.0% (String), and 3.2% (Lookup) for the respective scenarios. The memory usage stays at 5% for all five scenarios. We conclude that the firewall resource usage is very low, even on our modest hardware. This shows our firewall can handle traffic loads which are considered high for SH networks. Additionally, it is very lightweight and can be deployed on virtually any home router supporting OpenWrt.

5.6 Discussion and related work

Our solution extends on existing concepts in the profiling of the network communication of SH devices, to provide a fine-grained anomaly-based security system. Nonetheless, it also raises additional concerns. In this section, we discuss various aspects of our solution by comparing it to existing related works, and propose insights about potential future improvements.

5.6.1 Lightness of the security system

Numerous authors have developed security systems for IoT or SH networks. The vast majority leverages ML models. A typical example of such ML-based systems is *Passban IDS*, developed by Eskandari *et al.* [115]. It is an anomaly-based IDS which can be deployed on gateways with commodity hardware. The IDS proposed by Pashamokhtari *et al.* [152] employs sequential ML models that are specialized on subsets of traffic features describing the expected traffic of SH devices. They employ SDN to dynamically adapt the traffic forwarding rules.

ML-based IDSes often require heavy resources as well as extensive training data. Our approach is inherently opposite: by being based on a simple

model of the intended device network behavior, it only requires device profiles as input, and is very lightweight. Other researchers have followed a similar approach. Barrera *et al.* [21] defined a profile format which was even simpler than ours, as it did not support interactions involving multiple dependent patterns. This simplicity allowed them to leverage the plainer firewall IPTables. Their solution is more lightweight than ours, but at the cost of support of interaction traffic patterns.

While we designed a new profile format inspired by MUD and deployed a firewall on top of it, Hamza *et al.* [345] directly leveraged MUD profiles as their model of allowed traffic. To provide security based on those, they converted the ACLs in the MUD profiles into SDN rules. As they use MUD directly, their security system suffers from MUD's shortcomings, as described in Section 5.2.3. Besides, since they leverage SDN, the network must support it and include an SDN controller. This makes their solution less portable than ours, which leverages a default Linux kernel software.

5.6.2 What types of attacks can/cannot be blocked?

Our approach focuses on attacks with an observable impact on the network behavior of the affected devices. Since the firewall monitors all its interfaces, it can also block attacks *from* SH devices, such as outgoing DDoS attacks or network scans. Our firewall performs lightweight Deep Packet Inspection (DPI), limited to unencrypted protocols. For example, it is able to match the protocol fields of DNS or HTTP traffic. Apart from that, payload is not further analyzed, which means that attacks that do not look suspicious on metadata level or traffic statistics level (such as an SQL injection attack) cannot be stopped.

Our firewall cannot inspect protocols fields of encrypted protocols, such as the URL in HTTPS requests. Note, however, that many IoT devices use clear text protocols due to resource constraints or because the manufacturer is shy of the associated effort (updating certificates, etc.). Furthermore, previous research has shown that the headers and metadata of network traffic can provide sufficient information for detecting many types of attacks [145, 152].

Data exfiltration attacks targeting privacy in Smart Homes can be blocked depending on their nature. Apart from blocking data transfers to unknown destinations, the firewall can limit the rate, duration, and size of network activities. Therefore, even if the destination is generally trusted, the firewall can block data transfers that deviate from the expected behavior, such as a smart speaker with a microphone sending more data than expected.

Traffic related to user events (e.g. activating a device) might be indistinguishable from traffic not initiated by the user (e.g. keep-alives). In that case, our firewall might prove to be imprecise, if a specific user event must

be blocked whilst other traffic should still be accepted. Nonetheless, for most devices, this type of traffic *can* be distinguished, thanks to different packet signatures, or an identifiable periodicity [346].

For this study, we focused on devices communicating via the typical TCP/IP stack. Conceptually, our approach could be extended to other SH-related protocols, such as the ones described in Section 2.1.3 (e.g. Zigbee or Lo-RaWAN). Nevertheless, this would require substantial development work, in order to add the necessary hardware and the protocol decoding logic.

5.6.3 Level of abstraction of profiles

Our profile format allows expressing the communication patterns of SH devices by specifying the traffic associated to them. The latter can be described with attributes pertaining to protocols sweeping through the OSI layers, from the data link to the application layer. The patterns are thus described in a low-level fashion, using the very packets which are the building blocks for network communication. The level of abstraction of our solution is quite low. The advantage is that our profiles can be used as explicit descriptions of the traffic which should be accepted by a network security system, which is precisely what we did with our pipeline producing firewall configuration files. However, in practice, a device might issue diverging patterns to perform the same action, e.g. a smart plug turning on, depending on the situation. For the sake of exhaustiveness, the profile for said smart plug should include all patterns variants, which substantially lengthens the profile, and renders the process of creating the profile more tedious. This gets even worse when considering interactions involving a sequence of events, as they might suffer from a combinatorial explosion of patterns. Take two subsequent events which respectively correspond to m and n different traffic patterns: the interaction involving the sequence of those two events is described by a total number of $m \times n$ patterns.

To make up for those drawbacks, one could envisage describing the communication patterns in a more abstract way. Instead of specifying all related network packets, one could simply list the individual events taking part in the interaction. Doing so fixes the aforementioned drawbacks: each event is specified as is, which means only one variant in the profile, instead of the multiple ones due to varying traffic. Consequently, sequences of events need only be specified once, instead of $m \times n$ times. Hu *et al.* [346] followed this approach and proposed BehavIoT, a system able to model cross-device interactions with their constitutive events, abstracting the underlying traffic. Their system, however, only serves for modeling, and was not designed as a security system capable of blocking events.

Sun *et al.* pushed it further with their system *D-interact* [27] which mod-

els cross-device interactions, then identifies potential attacks as well. In their work, they consider two channels which enable interactions. The first are SH automation platforms, which is the one we consider as well. The second are physical properties (*e.g.* room temperature) which can serve as intermediaries between two events, if one modifies it and the other reacts to the new value. Interactions are modeled as a directed graph: nodes represent SH events and physical channels; an edge from node *a* to node *b* embodies an interaction where *a* is followed by *b*. What they consider as “attacks” is any path of length greater than two, which might manifest an unwanted sequence of interactions. While their modeling covers a broad scope of interactions and is sound relative to a real-world SH installation, considering any path longer than two as an attack might be a too aggressive security measure, as those paths might also be desired by the SH user. Besides, their work is again not designed to block events actively, but only as a preliminary model to detect potential vulnerabilities *a priori*.

This more abstract approach to profiles raises a new practical challenge: how to block communication, not based on its actual traffic, but on the more abstract description of its SH events? Such events have meaning only in the context of a SH platform, whether a device-specific or a more generic home automation one. While device-specific apps are restrictive in that they support only the devices they were designed for, a generic home automation platform is, generally, able to support a wide range of devices from different vendors. A concept of a solution could be to instrument such a platform to detect the ongoing event and issue the verdict based on the profile’s content. This requires a platform which is sufficiently open to be instrumented as such. While a commercial platform such as SmartThings [299] would be too limiting, there exist open-source home automation platforms which can be easily augmented with custom capabilities; the most popular being *Home Assistant* [347] and *openHAB* [348].

The motivation behind this work is to provide network security. Our firewall is designed to allow or block traffic, based on the input device profiles. Improving network security is one concrete way to enhance other, more abstract guarantees, such as privacy or safety. Other works followed a similar design approach by leveraging user-defined profiles, but not necessarily to provide network security *per se*.

Nowadays, SH installations may involve various SH apps and platforms, which must often cooperate to perform home automation apps. Interactions between those can be nerve-wracking to apprehend, even more to control. With EXPAT [349], Yahyazadeh *et al.* developed a system which enforces user expectations, regardless of the interactions and different platforms which come into play. Similarly to us, they designed a policy language able to express user-defined invariants involving the state of SH devices or physical

properties, which must stay satisfied no matter what. If an interaction is about to conclude on an undesirable state, it is prevented. Their approach works at a higher level of abstraction than ours, as it does not consider the network traffic, nor the actual events which are executed; the only thing that is enforced is the state of the SH itself. Strictly speaking, they do not consider security, but safety.

By definition, a SH network takes place inside its user's household. Usually, a household comprises several people, which might all wish to interact with the SH system. However, users might want to restrict functionalities, and prevent access to some of them to children or guests. Sikder *et al.* [350] proposed a platform-agnostic system which enforces precisely that. Again, they defined a policy language allowing users to describe rules for controlling access to SH devices. Their work is positioned at a high level of abstraction; we argue that it could be improved by enhancing the granularity even more, by enforcing access control at the level of an event instead of a complete device. The core of their work is user access control, a facet of security which we did not consider in our work.

5.6.4 Creation and distribution of profiles

Device profiles must accurately describe the devices' communication patterns. A substantial question which was left out in this chapter is that of the creation and distribution of the profiles. In the context of this chapter, we created those manually. However, their usage scenario is to secure SH networks, an environment where, as pinpointed in Section 2.2.1, users are not necessarily knowledgeable about their operation. As such, users should not be responsible for the creation of their devices' profiles. Besides, as popular devices are potentially deployed by many users, it would be way more optimized to provide a centralized, trusted repository of profiles, which users can simply query for the ones related to their devices. This architecture was actually envisioned in the MUD standard [320]. Indeed, the standard states that devices emit a MUD URL, which can be queried to obtain the device's MUD profile from a server.

Now, the question is about who should create those profiles in the first place. A typical assumption, which we followed, inspired by related work, is that the profile is created by the device's manufacturer. The main advantage of this approach is that the manufacturer knows the inner workings on the device, and can therefore create a profile which covers the device's network behavior exhaustively. However, numerous examples show that manufacturers tend to implement non-required behaviors in their devices; their rationale can be advertising services, collecting operational statistics, or even covertly extract sensitive user data [330]. In order to only allow communication pat-

terns required for the device's operation, profiles could be created by experienced users (such as us, in the context of this study). This process is quite tedious and error-prone, though. Numerous researchers have designed systems which automatically extract such profiles from captured network traffic from the devices. An in-depth analysis of existing approaches will be undertaken in Section 7.1.

As we demonstrated throughout this chapter, profiles should be more expressive, and include temporal dependencies between different patterns to provide a more realistic security. While this improves expressiveness and granularity, it makes it way harder to design a system which could automatically extract profiles simply from packet captures without any other information. The aforementioned system BehavIoT [346], is able to model the behavior of a complete SH deployment. However, it does so by applying statistical and ML models on the traffic recorded from within the Smart Home. The resulting model is therefore inherently tied to the network it was applied to, and is hardly applicable to other deployments where traffic patterns may diverge. Obtaining the model for a complete SH network is, in the general case, not as trivial as simply merging the profiles corresponding to the devices present in it. Nevertheless, to push automation even further, one can envision a system where devices are automatically detected inside the SH environment; numerous research works have been conducted in this direction [20, 335, 184].

The purpose of the recent technology of the blockchain is, broadly, to store information which cannot be tampered with [351]. The Connectivity Standards Alliance (CSA), *i.e.* the organization which issues certifications to products complying with the Matter protocol, maintains the Distributed Compliance Ledger (DCL) which tracks the status of Matter-certified devices in an unalterable fashion [78]. It also occurs as a promising solution to securely and publicly store MUD or analogous profiles. Liu *et al.* [352] first leveraged a blockchain, namely the open-source Hyperledger Fabric, to store MUD profiles as immutable smart contracts. Rath *et al.* [353] pushed the concept further by designing a complete pipeline which automatically produces a MUD profile based on the device's network traffic, stores it in a blockchain, and implements rule enforcement in the network.

Finally, the behavior of devices might change over time, usually due to a firmware update. In that case, its profile, and the associated firewall configuration files, must be updated to reflect the new behavior. During our experiments, we had to do this for one of the devices.

5.6.5 Extensions to the MUD standard

As discussed in Section 5.2.3, the MUD standard suffers from multiple shortcomings. Several authors have tried to overcome those by proposing extensions to the base MUD format. Some were already mentioned in the aforementioned section.

Traffic statistics Lear *et al.* [323] proposed an RFC draft extending MUD with the capability to specify the traffic rate, more precisely the number of packets or bits per second. Our profile language supports their fields, besides other traffic statistics such as the total number of packets and the duration of the traffic flow.

New protocols Reddy *et al.* [354] authored another standardized IETF RFC, bringing support for TLS features to MUD. The additional fields include the encryption, signature and key exchange algorithms, which allow enforcing the use of recent algorithms instead of old, unsecure ones. A master's thesis [355] extending the project presented in this chapter similarly added support for the TLS version and handshake type in the device profiles, and the associated logic in the firewall.

Matheu *et al.* [324] extended MUD with the capability of matching application-layer protocols by their name, as well as the number of concurrent connections using a specific protocol and the accessed resources. Our work pushes this concept further by allowing matching traffic based on specific fields of said protocols, instead of only the protocol itself.

Others Morais and Farias [356] added *Malicious Traffic Description* fields to MUD in an effort to integrate signature-based intrusion detection. Finally, Matheu *et al.* [357], in a later work than the aforementioned one, leveraged the Medium-level Security Policy Language (MSPL) [358] to provide data privacy, resource authorization, and channel protection capabilities to MUD.

5.7 Conclusion

Modern SH installations exhibit complex communication patterns caused by the interactions of the deployed devices. Simple firewalling rules ignoring these interactions leave opportunities to a number of attacks.

Taking inspiration from the IETF's MUD standard, we defined a new language for communication profiles of IoT devices. Contrarily to MUD, the language is expressive enough to describe communication patterns involving multiple devices and their interactions. We designed a lightweight firewall that uses the profiles to enforce communication policies in SH networks. We

performed experiments with various types of IoT devices and showed that the firewall is effective in blocking unwanted network activities, while remaining transparent to the devices and the users.

This work was presented at the 2024 IFIP Networking Conference [2]. The code related to this work has been released as open-source project. Since the publication, two master's theses have worked on improving and extending it. Laurent [355] added support for the TLS and MQTT protocols to the profiles and firewall. Gallez and Zhang [359] introduced various improvements: traffic matching based on the time, similar to cron [360]; support for pattern loops in a profile; tool which extracts simple patterns from a given PCAP file uploaded via a user-friendly Graphical User Interface (GUI).

In-depth Profiling and Robustness Assessment of Smart Home Events

6

The previous chapter (5) described a profile format expected to exhaustively describe the communication patterns a SH device might issue. With such a format defined, a logical next step is to provide a methodology to automatically extract the communication patterns which could be used to populate the profile. While substantial research effort has been devoted to this topic, existing works kept their analysis at a superficial level when profiling the network communications of SH devices.

In this chapter, we tackle Research Question 2 by presenting a framework able to extract the communication patterns issued when SH devices perform their usual actions. Contrarily to existing works, we perform an in-depth analysis by selectively blocking already encountered patterns in order to make new ones appear. This workflow successfully extracted patterns systematically missed by related work.

We argue that such patterns, which we dub as *hidden*, are a good measure of a SH device's robustness: indeed, they allow the transmission of messages even when some traffic cannot succeed. As such, we also address Research Question 3, and perform an extensive assessment of said robustness over the covered devices, by defining and computing empirical metrics.

Two articles are associated with the study presented in this chapter. An early version has been presented at the 33rd ICNP in 2025 [3]. A revamped, extended version of the work is currently in preparation for submission to a journal.

6.1 Introduction

As introduced in Section 2.3, the ultimate purpose of a SH installation is to enhance the convenience of the user, by enabling home automation, and allowing the user to define routines, *i.e.* sequences of actions from one or multiple devices, which achieve a specific goal.

A SH environment is a network comprising numerous devices interacting with each other. Because the user-defined routines can involve virtually any

device, they all are of equivalent importance; if only one of them fails, the whole system might be disrupted. Therefore, SH devices should be robust, *i.e.* they should be able to function properly in any situation. This is exactly the motivation behind this work: our objective is assessing the robustness of SH devices. The term “robustness” being quite vague, it can be understood in countless ways. We focused on a very practical way to do so, tied to the network communication patterns issued by a device: if the default pattern issued to perform a given action fails, different backup patterns should be performed to ensure success. Such a pattern, which occurs only as backup when the default one fails, is dubbed as *hidden*. In this study, we aim to discover and describe all network patterns issued by SH devices, whether in ideal conditions, or whenever some patterns cannot succeed.

A common factor shared among all subdomains of the IoT, already discussed in Section 2.2.1, is the limited computing resources available on the devices. Contrarily to general-purpose devices (*i.e.* computers, smartphones), IoT devices are intended for a definite purpose. Their electronics are often specifically designed, and its resources limited to lower the costs. However, this also means that manufacturers must find a compromise on which functionalities to include on the limited hardware. Notoriously, to stay economically competitive, they tend to choose expanding the user-visible functionalities, neglecting transversal aspects such as robustness or security [361, 17]. While those aspects are not directly visible by the users, they remain of utmost importance: neglecting them might hinder the device’s correct operation, or even render it unusable. Ironically, this compromise impedes the initial incentive behind the design of SH systems, *i.e.* user convenience. Indeed, if a device stops functioning due to a lack of robustness, *e.g.* in case of network instability, the user-defined routines will fail, hindering the very purpose of the Smart Home altogether.

A common assumption regarding the behavior of SH devices is that they usually exhibit simple and predictable network communication patterns [20] under ideal conditions. Consequently, researchers have designed solutions to express the network behavior of such devices in a compact form, a so-called *profile*, as presented in Section 5.2.2. The MUD format, being an IETF standard, acts as the flagship of such profile formats, even if albeit admittedly relatively limited [362, 363, 2]. Several methods have been presented in the past to automatically create the profile for a device [364, 23, 365]. For this purpose, its network communications are observed over a certain period of time, in the wild or under laboratory conditions. Typically, relatively complex experimental setups are required [184, 346], due to the variety of possible interactions with the device. For instance, interactions with a light bulb might include turning it on and off, changing its brightness or color, etc. Data mining and analysis techniques are then used to extract compact representations

of the network traffic and associate them with the device, or even with specific device events.

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

In this chapter, we present a framework, mostly automated and requiring little prior configuration, to comprehensively uncover the communication patterns of a SH device. For each type of interaction with the device, it first observes the traffic flows that occur in an unconstrained network and builds a set of flow descriptors from them. Such descriptors contain Internet and transport-layer information, as well as selected, unencrypted, available application-layer data, such as domain names. The system then iteratively blocks flows corresponding to certain descriptors and repeats the interaction with the intent of making new flows appear. The algorithm selecting which flows to block and the sequence of blocking experiments is not straightforward. Our aim is to discover as many hidden flows as possible but keep the number of experiments under control. To do this efficiently and not end up in an infinite loop where the same patterns appear over and over again, we store the found descriptors in a tree that we traverse using a breadth-first strategy. Besides, we propose two different *ad hoc* pruning heuristics to stop traversing the tree. The result is a set of *multi-level*, tree-shaped profiles (one for each type of interaction) for the device that describe the device's default and alternative communication patterns that we then explore to assess the robustness of the device's network communications.

Afterwards, we conduct a multistage empirical evaluation of our framework, and of its resulting profiles. We start by defining and computing metrics which shed light on the nature of the traffic emitted by SH devices. For each studied interaction, we start by enumerating the different communication patterns at hand, then analyze in more detail what protocols are used, focusing especially on data pertaining to the DNS protocol. We articulate this evaluation around a comparison between the two pruning heuristics, eventually deciding which one fits the most. Additionally, we assess the robustness of the network communication of the covered devices. A first, rough measure

of the robustness is given by the number of *hidden* network patterns issued for a given interaction. We then refine the robustness assessment by taking into account the occurrences of success and failure of the interaction. We end up with a formula expressing the probability that an interaction will fail, based on the network traffic which has been blocked. Based on this expression, we evaluate the probability of failure when blocking all traffic pertaining to a given protocol, or any set of random traffic flows.

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

The contributions of our work can be summarized as follows:

- **We develop and implement a framework**, modular and easily extensible, which utilizes SH devices and **extracts multi-level profiles** for them, in a **mostly automated way**. The core of the framework is a new algorithm that, for a given device interaction, builds an event signature, *i.e.* the description of flows appearing for that interaction, and then iteratively blocks previously observed flows, until the interaction fails or the process is stopped by the selected pruning heuristic.
- **We apply our framework** to a testbed network comprising 11 off-the-shelf devices, and generate multi-level profiles for 38 unique events, using two different pruning heuristics. We extract a total of 373 traffic flows (119 more than in previous work), of which 115 were “hidden”, *i.e.* not part of the default communication patterns of the tested devices, accounting for over 30% of the total number of flows, representing the fraction that traditional techniques are likely to miss.
- **We perform a holistic empirical evaluation** of the framework, by computing various metrics over the resulting data. Those give an insight on the communication behavior of SH devices. The analysis is structured along a comparison between the two pruning heuristics, and is concluded by a discussion on the better heuristic to use.
- **We assess the robustness** of the employed devices, by computing robustness-related metrics. Additionally, we model the probability that

a device interaction fails, based on the traffic that is blocked. Our main observation is that 28 out of the 38 triggered interactions provide at least one backup communication strategy if a default communication pattern fails; a promising result regarding the robustness of the devices' network communications.

- **We publish the source code** of our signature extraction algorithm at <https://github.com/smart-home-network-security/signature-extraction>, and of our experimental framework, **including the captures of the testbed's traffic** at <https://github.com/smart-home-network-security/smart-home-hidden-communication-patterns>.

The rest of this chapter is structured as follows. We describe the problem and the scope of our work in Section 6.2. In Section 6.3, our methodology is explained in detail, and Section 6.4 presents our testbed of real-world devices on which we applied the framework, which is a subset of that described in Chapter 4. Subsequently, we perform deeper analysis of the results. In Section 6.5, we evaluate the framework empirically, by computing various metrics over the resulting data, *e.g.* occurrences of protocols in the extracted patterns. The pruning heuristics are also compared. Following the main incentive of this chapter, Section 6.6 assesses the robustness of the studied devices, by computing metrics based on the discovered *hidden* patterns, and expressing the probability that a device interaction fails. To wrap up the evaluation, we compare our results with related work in Section 6.7. Section 6.8 presents more related work and positions them compared to ours. Ultimately, we conclude the chapter in Section 6.9.

6.2 Problem statement and scope

Before delving into the details of our solution, this section will specify the context in which it is positioned, and the scope of the problem we aim to tackle.

An example SH setup is depicted on Figure 6.1. It involves a smart lamp that can be controlled by a smartphone application (called *companion app* in the following). The network communication issued by or towards the lamp can be broadly classified in three categories [366, 346]:

- **Triggered communication** This type of communication occurs when a user actively interacts with the device. In the case of a smart lamp, such interactions can include toggling the lamp, changing its brightness or color, etc. By design, it can occur whenever the user decides.
- **Periodic communication** As its name suggests, this type of communication encompasses traffic which is issued regularly, with fixed time

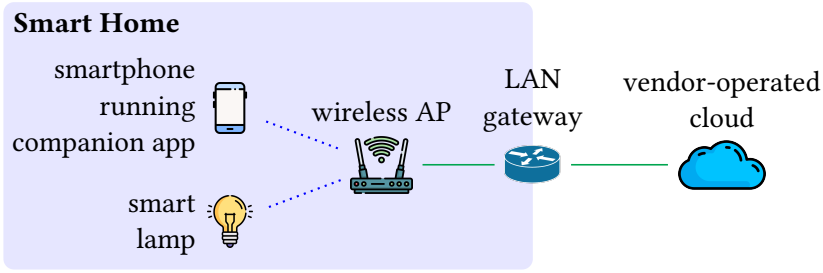


Figure 6.1: Example SH network involving a smart lamp

intervals. Its purpose is various, ranging from exchanging keep-alive messages with another (remote) host, local discovery of other devices, or checking for updates.

- **Aperiodic communication** The last category comprises all communication which fits in neither of the two aforementioned ones, *i.e.* spontaneous (not triggered) communication which does not follow a periodical schedule. Examples of such patterns are more sparse: Hu *et al.* [346] report they account for less than 1% of their recorded traffic.

In all three cases, the network patterns issued are highly device specific. The device might communicate directly with the companion app in the local home network, or do this via a vendor-operated cloud server, the latter making it also possible to control the device from outside the home network and to receive firmware updates. In fact, many devices on the market have both local and remote communication capabilities.

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

The solution that we will present in the next sections requires the collection, inspection and filtering of packets sent to and received from the SH device and the device running the companion app. If the traffic is unencrypted, we leverage its payload. We do not analyze or modify the device firmware, the companion app, or the server side software. We also do not try to decrypt network traffic, therefore we do not consider information in encrypted payload. However, numerous research has shown that relying on unencrypted

information and packet or flow metadata, can already provide sufficient information to accurately fingerprint IoT devices [23, 365, 367].

Concerning the category of periodic communication, a possible approach to profile such activities is to passively monitor the device over a longer period and extract the traffic using periodicity models [346]. Our approach does not pursue such communication activities further, as we focus on the more complex case of interaction-triggered communication patterns. The class of aperiodic communication is, by design, impossible to characterize by itself; the most promising way is to characterize and filter out both other categories.

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

Similarly to related research works, we only consider IP traffic, either wired or wireless. Other protocols exist for SH devices, such as Zigbee [368] and Thread [77] over IEEE 802.15.4 [369], or the proprietary Z-Wave [370]. Such protocols usually rely on a gateway or hub to connect to the local home network. For devices using these protocols, we profile the gateway's IP traffic. Devices that rely on long-range communication technologies, such as 5G or LoRaWAN [371], are not in scope since their traffic cannot be locally filtered or blocked.

6.3 Description of the framework

In this section, we will describe the framework we designed to extract communication patterns of SH devices. We will start with a big picture overview, then delve into the details of each building block.

6.3.1 General workflow

Figure 6.2 gives an overview on the workflow of our methodology to obtain a profile for a SH device under a certain type of interaction. An interaction can require the usage of a triggering source, such as a companion app installed

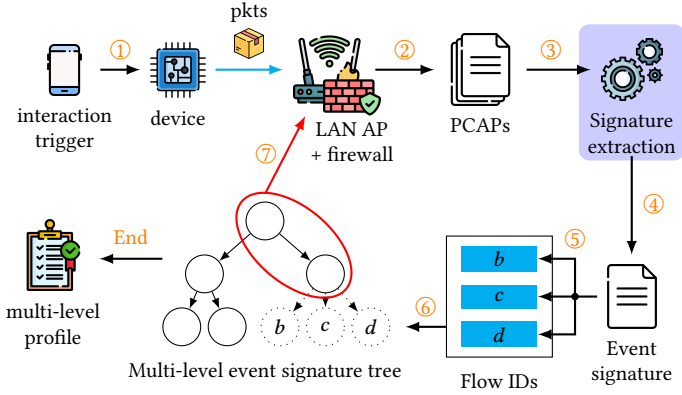


Figure 6.2: Overview of the profiling workflow

on a smartphone. In practice, a SH device may support different interactions (e.g. for a smart lamp: switching on/off, changing its color, etc.) and triggering sources, and the workflow described here must be repeated for each of them. The workflow consists of multiple steps summarized below. Besides initial manual configuration in Step 1, the rest of the workflow is automated.

The first steps consist in performing the interaction with the device under investigation (6.3.2) and recording the network traffic (6.3.3). To this end, we connect the SH device (or, for non-IP devices, their gateway or hub) and the device running the triggering source to a wireless (resp. wired) LAN and capture the traffic on the AP (resp. switch). The framework is agnostic to the device’s actual communication medium, *i.e.* wired or wireless. We call *event* the interaction along with its resulting device state. The interaction is performed multiple times to alleviate experimental errors.

The traffic traces obtained are then filtered: only the ones pertaining to successful interactions are kept (6.3.4). An *event signature* is extracted from the remaining traces (6.3.5). It consists of a set of traffic flow descriptors, called *Flow IDs* in the following, that describe the relevant features of the bidirectional flows observed in the traces, and therefore associated with the event. A Flow ID contains network and transport layer information, such as host addresses and port numbers, as well as application layer information (when available). The extracted Flow IDs are stored in a tree-shaped data structure that allows tracking which Flow IDs have already been observed.

In the next steps, the framework selects a Flow ID from the tree (the next one in BFS processing order), configures a firewall running on the switch or AP to block packets matching that ID (6.3.7), and repeats the aforementioned steps. The idea is that by deliberately blocking specific traffic patterns associated with the interaction, the device and/or app will try alternative patterns.

The exploration of the tree, and thus the experimental workflow, are stopped whenever an *ad hoc* condition is met (6.3.8). The final result is the profile of the device for the investigated event. It takes the form of a tree, where each node is a Flow ID linked to the event, and the path from the root towards a given node gives the set of ancestor Flow IDs which were blocked to let the current Flow ID appear. We define the Flow IDs initially occurring without any traffic blocking required, *i.e.* the ones appearing in the tree at the first level (as children of the root) as *first-level* Flow IDs. Conversely, the ones appearing as a result of traffic blocking, *i.e.* at a level of two or deeper, are dubbed *hidden* Flow IDs. The latter are the communication patterns which can be discovered only through our multi-level approach, and therefore missed by state-of-the-art techniques.

6.3.2 Interaction trigger

To be able to systematically profile SH events, we need a strategy to automatically trigger the surveyed event. To do so, we instrument a smartphone running the companion app able to issue commands to the device. In practice, we used a smartphone with the Android OS, such that we can use Android Debug Bridge [312] to script (user) interactions with the phone; in our case, touch screen inputs performing the event. Prior manual work was required to configure such screen touches with the correct coordinates for each app and each event.

Such an interaction can fail for a variety of reasons, potentially due to the nature of our experiments, or circumstances outside our control. To obtain more reliable results, each interaction is repeated in a batch of m iterations.

6.3.3 Event traffic capture

In parallel to the event execution as explained in the previous section, network traffic is captured, with the intent of retrieving the traffic issued by the event's operation. The duration of the traffic capture must be sufficiently long to ensure all the event-related traffic has been captured. While this is impossible to ensure *a priori*, we can take a conservative approach by configuring a duration deemed long enough, used in similar context by related work [23, 367]. Resulting traffic captures are stored in the PCAP format.

As we want to pinpoint only the traffic related to the event, we apply traffic capture filters, and only keep packets exhibiting the device's MAC or IP addresses, or the smartphone IP address.

Additionally, we want to keep only the traffic which is semantically meaningful for the event. Therefore, control plane packets are filtered out, including all ARP, ICMP and DHCP traffic, as well as TCP and TLS handshake pack-

ets, except, for the latter, Client Hello packets which can bear DNS data in the Server Name Indication (SNI) extension field.

6.3.4 Event success verification

As our experimental workflow, by design, tampers with the issued network traffic, events can fail, due to the required traffic being blocked. An event failure means the exploration of the behavior tree must be stopped; as such, the framework provides a way to check for the event's success, similarly to Mandalari *et al.* [314]. While they compared the pixels between screenshots of the companion app, before and after the event's execution, we used an approach we deemed less error-prone, *i.e.* leveraging official or third-party Python libraries specifically designed to interact with the devices, *e.g.* python-kasa [316]. The approach is detailed in Section 4.4.2.2. It should be noted that, while those libraries communicate with the device over the network, we filter out this traffic, to ensure the integrity of the measurements. If no such library exists, we fall back to screenshot-based verification, as the aforementioned related work, but, instead of comparing the screenshot's pixels, we compute the SSIM [315] between both images, and consider them equivalent if this measure is above an empirically defined threshold. The latter is specific to the interaction and companion app at hand. We consider a batch of m event iterations for further processing if more than half of the events were successful, according to this verification strategy. The outcome is a set of $m^+ \leq m$ traces serving as the input for the subsequent step.

6.3.5 Event signature extraction

The core of the workflow is an algorithm that we developed which takes a set of PCAP files containing event-related network traffic, and extracts the network flows which are observed upon the event's execution, which we dub the *event signature*. This section will detail this algorithm.

The input is m^+ PCAP files, each containing traffic related to a successful event execution, as well as potentially other, parasite traffic. It is fed into a pipeline which outputs the signature resulting of one batch of event executions; its steps will be detailed in the following.

6.3.5.1 Replacing IP addresses by domain names

Over the Internet, hosts do not usually address their communications with direct IP addresses; they tend to use domain names. Those allow for more flexibility, as a single domain name can be routed to various addresses, to handle latency or load balancing. The DNS protocol is used to resolve domain names to IP addresses before issuing the traffic. For the same flexibility

reasons, we want our signatures to rely on domain names whenever possible, and fall back on IP addresses if the former are not available. To do so, our framework stores a table mapping each encountered domain name to its IP address(es). We implemented a two-stage mechanism to keep this table up-to-date during the experiments.

The first stage occurs before starting each batch of event executions. The framework retrieves the DNS table of the LAN gateway, and uses it to populate the framework's table initially. Then, upon reception of any packet bearing DNS data, the table is updated with the new mapping. Such packets are DNS responses or TLS Client Hello messages including the SNI extension.

For the addresses that could not be mapped during the experiments, we tried using reverse DNS lookups *a posteriori* to find their domain name. However, since most of them were hosted at big cloud providers such as AWS, the only data we got from that was domain names in the AWS range.

6.3.5.2 Aggregating packets to Flow IDs

We define the concept of *Flow ID*, which is an augmented version of the typical network 5-uple representing communication flows. A Flow ID is an abstract representation of a set of packets relating to a bidirectional communication. It gathers all fields which carry semantic meaning to characterize this communication, namely the following fields:

- Hostname of both communicating hosts, dubbed *source* and *destination*. As per the previous section, we consider domain names whenever possible, and fall back to IP addresses if no domain name is available. As the Flow ID is bidirectional, which one is the source (resp. destination) is not relevant. The Flow ID's source is considered as the source of the first packet observed for this flow.
- Transport layer protocol, *i.e.* TCP or UDP.
- Ports used for transport-layer communication. We only consider ports which identify a service, whether well-known ones (*e.g.* 53 for DNS) or empirically discovered vendor-specific ones. The ports are tied to their related hosts.
- Various application-specific data for the supported unencrypted application-layer protocols. Among others: query type and name for DNS; method and URI for HTTP; message type, code and URI for CoAP. Linked fields are aggregated in the same Flow ID, *e.g.* a DNS response is grouped with its query.

6.3.5.3 Extracting the relevant Flow IDs

For each event iteration, the Flow ID aggregation produces a set f_i of Flow IDs, with $1 \leq i \leq m^+$. The event signature s is built by taking the intersection of all these sets:

$$s := \bigcap_{i=1}^{m^+} f_i \quad (6.1)$$

By only keeping the intersection, we make sure to only keep the Flow IDs deterministically associated with the event, and filter out other parasite flows, such as periodic or sporadic messages.

6.3.6 Event signature tree definition

The event signature tree is a *rooted, directed tree*, defined as T :

$$T := (N, E, r, \ell) \quad (6.2)$$

where:

- N is the set of *nodes*, which represent the positions in the tree.
- $E \subseteq N \times N$ is the set of *edges*, directed from a *parent* node to a *child* node. Let $n_a, n_b \in N$: an edge directed from the parent n_a to the child n_b is defined as the ordered pair (n_a, n_b) .
- $r \in N$ is the *root* node, *i.e.* a node with no parent.
- $\ell : N \setminus \{r\} \rightarrow F$ is a function which assigns to each node, except the root, a Flow ID value from the set F , the set of all possible Flow IDs.

Each node of the tree carries data characterizing one Flow ID, except for the root node, which does not carry any data, and is just used as a structural starting point. This structure is constructed along the complete profiling of one event. It is used to keep track of the Flow IDs already seen and blocked.

For each node n , the complete process is repeated: the event under scrutiny is executed, and the network traffic is recorded. If the event was verified as successful (as explained in Section 6.3.4) the corresponding event signature s_n is computed from the recorded traffic using Equation 6.1, and each Flow ID contained in s_n is added as a child of the node n . If the event failed, traversal of the tree is stopped in this branch.

The tree is explored in a Breadth-First Search (BFS) fashion [372], *i.e.* all nodes at a given depth are processed before proceeding to the next level.

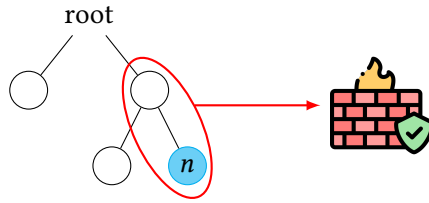


Figure 6.3: Firewall configuration based on Flow IDs contained in the tree.
The currently processed node (n) is colored blue.

6.3.7 Flow ID firewalling

Besides the root node, the processing of each node starts by configuring a deny-list firewall. The configuration process is illustrated on Figure 6.3. We define the *ancestors* of a node n as the set of nodes on the path from the root (excluded) towards n . The firewall is configured to block traffic exhibiting the same characteristics as the Flow ID information contained in the current node and all its ancestors.

In the previous chapter (5), we described an allow-list firewall providing the required matching capabilities. For the work described in this chapter, we implemented a deny-list variant of this firewall: it blocks the traffic matching the fields given as input. As the allow-list version, we released this tool as open-source¹.

The capture of event-related traffic is performed while the firewall is active, and the signature extraction algorithm is applied on the resulting traffic. However, since the traffic is recorded before the firewall is applied on it, the event signature might still contain a Flow ID which was blocked, *i.e.* one which is represented by the current node or one of its ancestors. To overcome this caveat, the exploration of a tree branch is stopped whenever encountering a node bearing Flow ID data equivalent to one of its ancestor nodes. This prevents endless exploration of a tree branch which would contain the same Flow ID over and over again.

6.3.8 Tree pruning heuristics

Our analysis considers the surveyed devices as opaque boxes: it does not extract information from the device's or its companion app's firmware; it only leverages the device's network traffic and state, and the information given by the app's GUI. Consequently, knowing whether the exploration of an event signature tree will stop, considering it is only stopped at nodes rep-

¹<https://github.com/smart-home-network-security/firewall-blocklist>

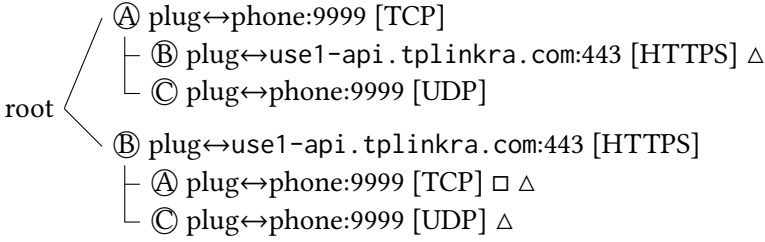


Figure 6.4: Tree pruning heuristics (considering BFS traversal).
Nodes marked with □ are pruned when applying *path pruning*.
Nodes marked with Δ are pruned when applying *node pruning*.

resenting an event failure, is analogous to the halting problem [373], and is therefore theoretically undecidable. Indeed, there can always exist very specific cases, issuing traffic patterns not previously seen, occurring only when cherry-picking certain Flow IDs to block. To ensure our framework runs in a finite and acceptable time, we provide two heuristics defining nodes upon which the tree exploration will be pruned, which will be described in the following.

6.3.8.1 Path pruning

The first heuristic is designed to account for equivalent sets of firewall rules. As all specific rules are blocking, and the default one is accepting, the activation order of the firewall rules is not relevant: a firewall’s ruleset containing two rules r_a and r_b is equivalent to one containing the rules r_b and r_a ². Therefore, it is unnecessary to apply a ruleset intended to block a set of flows, if, previously, an equivalent ruleset was already applied. We dub this heuristic *path pruning*, as it will prune exploration at a specific node, if a path equivalent to the one composed of itself and its ancestors has already been processed. Figure 6.4 depicts the node pruned when applying *path pruning* with □.

6.3.8.2 Node pruning

A more “aggressive” heuristic stems from empirical observations conducted with the TP-Link HS110 smart plug [284]. We observed that, for any two nodes with the same Flow ID, their children were always identical. Based on this observation, we define the *node pruning* heuristic: a node is pruned simply if an equivalent node, *i.e.*, a node carrying the same Flow ID, has already been processed. Figure 6.4 depicts the nodes pruned when applying *node*

²Note that this is only valid in our case, as the rule order might be important in a generic firewall operation.

pruning with Δ . This heuristic will dramatically reduce the size of the tree, but can miss nodes which would occur only after having blocked a specific set of Flow IDs.

6.4 Testbed and Example

In this section, we present our experimental setup, along with an example result we obtain by applying our framework on it. This setup is a subset of the complete one described in Chapter 4, and is different from the one used in Section 5.4.

6.4.1 Experimental setup

We apply our methodology to a controlled testbed network comprised of real-world devices, mimicking a typical SH network, depicted in Figure 6.5. We examine an array of commercial, off-the-shelf SH devices, including four power plugs, three cameras, and three light bulbs. The profiled interactions, per device category, are the following:

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

In a real SH network, a user could choose among various apps to control their devices, including the device’s official companion app (from its manufacturer or vendor), a third-party app, or a home automation platform which provides unified control over devices from different manufacturers. To cover such cases, for all interactions other than booting, we derive up to three usage scenarios:

- For all devices, we issue their interactions using their official companion app;
- For devices that support it, we also control them using the app of the popular home automation platform *SmartThings* [299];
- For the TP-Link plug and the Hue light bulb, we also experiment with other apps; respectively, *TP-Link Tapo* [302] (another app from the same manufacturer, targeted toward *Tapo*-branded devices) and *Hue Essentials* [305] (third-party app for Hue devices).

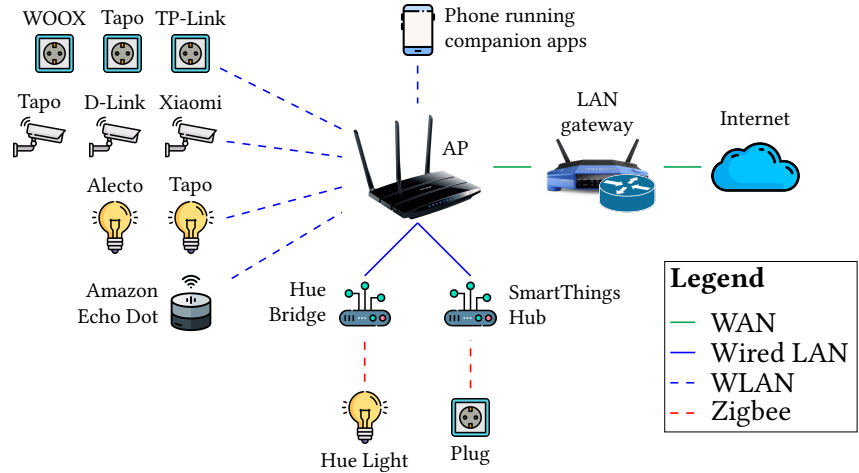


Figure 6.5: Subset experimental Smart Home network

The smartphone running the device-specific apps is a Google Pixel 9a [311] with the Android 15 OS. A LAN AP is present at the center of the network, to which all IP-enabled devices are connected, either wired or wirelessly. The AP is a TP-Link TL-WDR4900 [308], running the OpenWrt OS [120].

Device	App	Interaction	Event ID
Power plugs			
TP-Link HS110 [284]	/	boot	1
	Kasa Smart	toggle	12
	TP-Link Tapo	toggle	13
	SmartThings	toggle	14
SmartThings Outlet † [287]	/	boot	2
	SmartThings	toggle	15
Tapo P110 [285]	/	boot	3
	TP-Link Tapo	toggle	16
	SmartThings	toggle	17
Woox R5024 (Tuya) [286]	/	boot	4
	Tuya Smart	toggle	18
Cameras			
Xiaomi MJSXJ02CM [292]	/	boot	5
	Mi Home	stream	19
Tapo C200 [291]	/	boot	6
	TP-Link Tapo	stream	20
	SmartThings	stream	21

Device	App	Interaction	Event ID
D-Link DCS-8000LH [293]	/	boot	7
	mydlink	stream	22
Light bulbs			
Philips Hue White and Color lamp A60 † [374]	/	boot	8
	Philips Hue	toggle	23
	Hue Essentials	toggle	24
	Hue Essentials	brightness	25
	Hue Essentials	color	26
	SmartThings	toggle	27
	SmartThings	brightness	28
	SmartThings	color	29
Alecto Smart-Bulb10 (Tuya) [289]	/	boot	9
	Tuya Smart	toggle	30
	Tuya Smart	color	31
Tapo L530E [288]	/	boot	10
	TP-Link Tapo	toggle	32
	TP-Link Tapo	brightness	33
	TP-Link Tapo	color	34
	SmartThings	toggle	35
	SmartThings	brightness	36
	SmartThings	color	37
	Smart speaker		
Amazon Echo Dot (2nd generation) [298]	/	boot	11
		voice	38

Table 6.1: Devices in our testbed, with corresponding apps and interactions. † indicates Zigbee devices, for which the hub traffic was considered. The Event ID is a unique number assigned by us to each of the tested combinations of device, app, and interaction.

The complete device references are listed in Table 6.1, along with the corresponding instrumented smartphone apps and covered interactions. As explained, we only consider IP traffic, whether over Ethernet or Wi-Fi. Therefore, for the two devices communicating using Zigbee, we profile the traffic issued by their hub, *i.e.* the Hue bridge for the Hue lamp, and the SmartThings hub for the SmartThings Outlet, respectively. For the *boot* interaction experiments over those devices, we synchronously power-cycle both the device itself and its hub.

Our practical experimental parameters, inspired by related works *Ping-Pong* [23] and *IoT Athena* [367], are the following:

- Number of event iterations: $m = 10$;

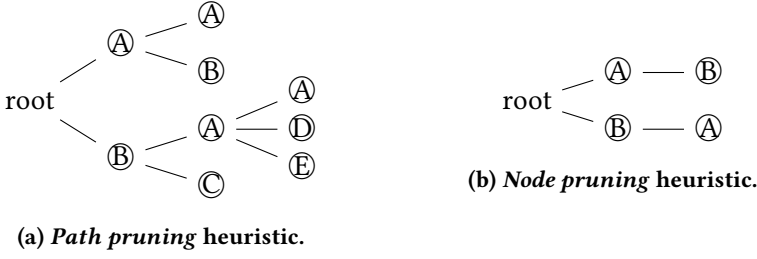


Figure 6.6: Event signature tree for the Tapo P110’s *toggle* event. The Flow IDs description is included in Table 6.2.

Ⓐ	device↔device-cloudgateway:443 [HTTPS]
Ⓑ	phone↔device:80 [HTTP]
Ⓒ	phone↔app-server:443 [HTTPS]
Ⓓ	phone↔app-cloudgateway:443 [HTTPS]
Ⓔ	phone↔app-tapo-care:443 [HTTPS]

Table 6.2: Description of the Flow IDs depicted on Figures 6.6a and 6.6b. When indicated, the port is associated with its host.

- Traffic capture timeout: $d = 20$ seconds (90 seconds for interaction *boot*);
- Duration between two event iterations: randomly chosen between 60 (90 for interaction *boot*) and 120 seconds.

6.4.2 Example signature trees

As an example of our framework’s results, we present the event signature trees generated for one device of our corpus, namely the Tapo P110 [285], when applying either the *path pruning* or the *node pruning* heuristics. This device being a smart power plug, it provides one interaction, *i.e.* toggling it on and off. For those experiment, we controlled the device using its official Android companion app, *i.e.* TP-Link Tapo [302]. The signature trees are shown on Figure 6.6; Figure 6.6a shows the one resulting from the *path pruning* heuristic, whilst Figure 6.6b shows the one resulting from the *node pruning* heuristic.

We observe that the node pruning heuristic is perhaps too aggressive, and that it can prevent specific hidden patterns to occur. Specifically, when the phone does not manage to communicate with the plug directly, it seems it uses a secondary path, through the cloud, by contacting the server responding to the domain name `*-app-server.iot.i.tplinkcloud.com`, abbreviated as *app-server* in Table 6.2. In the following, we will extend this preliminary

observation by comparing the outcome of both heuristics, when applied to our whole testbed.

6.5 Empirical evaluation of the framework and pruning heuristics

In this section, we apply our framework to all devices comprised in our testbed, and generate the event signature tree for all their events, sometimes triggered by different companion apps. We define various empirical metrics, each intended to assess a part of our framework's operation.

In the following, we will define each of those metrics in turn, and compute it over the resulting event signature trees. Due to the difference in behavior between the *boot* and the other events, we split the evaluation of the metrics between those two event categories. Contrarily to non-*boot* events, the success of a *boot* event cannot be defined in a straightforward way, neither be verified by a strategy as implemented in our framework (as detailed in Section 6.3.4). As such, for a *boot* event, the exploration of the tree exploration can never be stopped by an event failure, but only by the pruning heuristic. As the traffic related to *boot* events is very diverse, the framework's operation using *path pruning* would be intractable due to state space explosion. Therefore, for the *boot* events, we decided to apply the *node pruning* heuristic only. Nonetheless, all other events were evaluated using both *node pruning* and *path pruning*, and the results obtained with both heuristics were compared. More precisely, we computed three variant of all metrics, each counting the set of Flow IDs appearing in one of the three following cases:

- either when applying *node pruning* or *path pruning*
- only when applying *node pruning*
- only when applying *path pruning*

6.5.1 Exhaustiveness of the pruning heuristics

For starters, we want to assess if our framework is exhaustive (enough) when profiling an event's network behavior. More precisely, our objective is to evaluate which heuristic, namely *node pruning* or *path pruning*, is the most exhaustive. To this end, we computed two metrics, namely the total number of unique Flow IDs, and the number of Flow IDs extracted per tree depth, described in the following. Our initial assumption is that, since *path pruning* performs a deeper tree exploration, it will be able to discover more Flow IDs than *node pruning*, and therefore considered as more exhaustive.

Since *boot* events were only profiled using *node pruning*, we do not consider them for this metric.

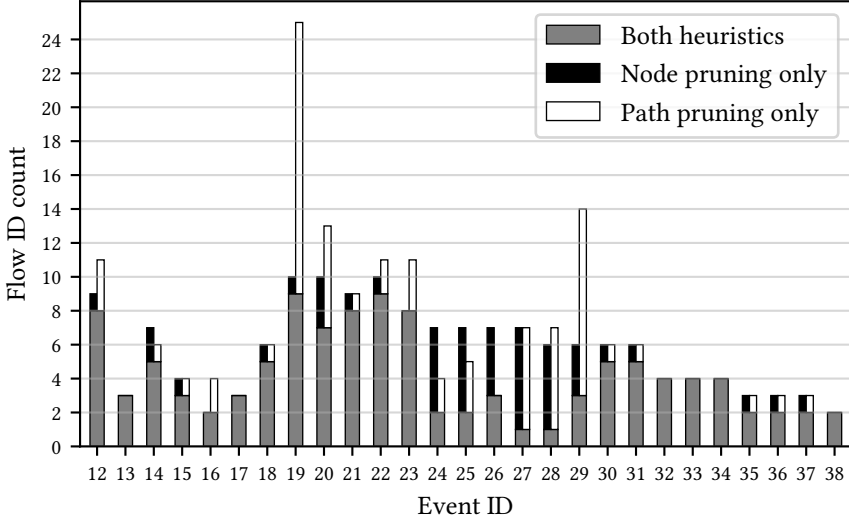


Figure 6.7: Total number of unique Flow IDs discovered, for both pruning heuristics

6.5.1.1 Total number of unique Flow IDs

The first and more straightforward is the number of unique Flow IDs present in the whole event signature tree. Considering a tree T as defined by Equation 6.2, we define U as the set of unique Flow IDs assigned to nodes in T :

$$U := \{\ell(n) \mid n \in N\} \quad (6.3)$$

where N is the set of nodes, and ℓ the function which assigns a Flow ID to each node. The number of unique Flow IDs is the cardinality of U , i.e. $|U|$. This metric helps comparing the two pruning heuristics, indicating the information lost due to *node pruning*'s more aggressive pruning.

The values computed over all events, except *boot*, split per pruning heuristic, are shown on Figure 6.7. To avoid cluttering the plots, instead of the device's and event's full name, the y-axis shows a numerical ID which corresponds to the one referenced in Table 6.1. The same strategy will be applied on subsequent plots where a similar comparison between the pruning heuristics is drawn.

Path pruning nearly systematically discovers more Flow IDs than *node pruning*, which confirms our initial assumption. The former can therefore be considered as the more exhaustive heuristic. The rare cases when the converse is true are nearly all related to the Philips Hue device, when controlled via the Hue Essentials app. In those cases, the Flow IDs discovered by *node*

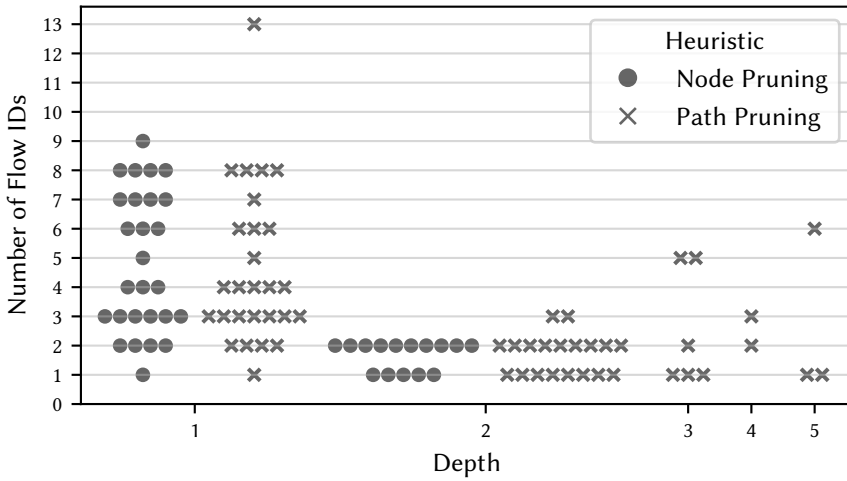


Figure 6.8: Number of Flow IDs discovered per tree depth, for both pruning heuristics

pruning only are mDNS messages used for service discovery in the LAN. It is possible that those messages are broadcast periodically during a given period, *i.e.* when we performed the measurement campaign using *node pruning*, then stop while we were doing so using *path pruning*.

6.5.1.2 Number of Flow IDs extracted per tree depth

Additionally, we want to see if the additional processing incurred by *path pruning* is really meaningful, *i.e.* if *path pruning* manages to find new, *hidden* Flow IDs at depths where *node pruning* has already stopped exploration. To this end, we plot the number of Flow IDs discovered per tree depth, for each of the two heuristics, on Figure 6.8. A Flow ID is *discovered* at a given depth d if it did not occur in the tree at a previous depth $c < d$.

Following our intuition, the *path pruning* heuristic is able to discover new Flow IDs at depths where *node pruning* has stopped doing so. By design, if *node pruning* does not discover new Flow IDs, it means it has stopped exploring the tree. We conclude that, even if *path pruning* requires significantly longer processing in order to explore deeper tree levels, it is still valuable as this allows discovering other *hidden* Flow IDs which would have been missed by *node pruning*.

6.5.2 Protocol usage

To gain additional insight about which protocols are used by SH devices, we compute, for each observed protocol, the number of Flow IDs which used said

Protocol	Events		Ratio
	All	<i>Hidden</i>	
Well-known application-layer protocols			
HTTP	5	0	0%
HTTPS	31	14	45%
DNS	82	47	57%
NTP	11	6	55%
SSDP	3	1	50%
mDNS	34	15	44%
Plain transport-layer protocols			
TCP	2	1	50%
UDP	6	1	14%

Table 6.3: Number of total & *hidden* Flow IDs per protocol, for *boot* events only

protocol. For a given Flow ID, we consider only the highest-layer meaningful protocol for which data was extracted; *e.g.*, for a Flow ID using HTTP over TCP, we only consider HTTP. We split the measurements between the two pruning heuristics. Then, in both cases, for each covered protocol, we compute two values: (1) the total number of Flow IDs regardless of their depth, (2) the number of *hidden* Flow IDs only. Additionally, we compute, per protocol, the fraction of all Flow IDs represented by the *hidden* ones, by taking the ratio of the latter over the former.

6.5.2.1 *Boot* events

The usage of protocols for *boot* events is shown on Table 6.3.

Besides plain UDP, for the protocols which are used for *hidden* Flow IDs, the ratio of hidden Flow IDs over all of them revolves around 50%. This means that those protocols are all used similarly when implementing alternative communication paths. Regardless of the protocol, and even in UDP's case, the strategy is generally to contact alternative endpoints or services.

6.5.2.2 Other, non-*boot* events

Regarding other events, the protocol usage numbers are given in Table 6.4. Here, we split the measures per pruning heuristic.

Compared to the values for the *boot* events, the fraction of protocols used by non-*boot* events' *hidden* Flow IDs is generally lower, around 30 to 40%, for the ones which do appear in *hidden* Flow IDs. A substantial part of the protocols simply do not appear in *hidden* Flow IDs. Akin to our previous results,

Protocol	Heuristic	Events		Ratio
		All	Hidden	
Well-known application-layer protocols				
HTTP	node pruning	9	1	11%
	path pruning	10	4	40%
HTTPS	node pruning	67	17	25%
	path pruning	100	31	31%
DNS	node pruning	5	0	0%
	path pruning	7	2	29%
SSDP	node pruning	1	0	0%
	path pruning	1	0	0%
STUN	node pruning	1	0	0%
	path pruning	3	2	67%
mDNS	node pruning	40	0	0%
	path pruning	23	7	30%
VoIP	node pruning	0	0	0%
	path pruning	5	5	100%
Ad hoc protocols based on transport-layer ones				
TCP/9999	node pruning	2	0	0%
	path pruning	2	0	0%
TCP	node pruning	10	4	40%
	path pruning	10	3	30%
UDP/7000	node pruning	6	0	0%
	path pruning	6	0	0%
UDP/9999	node pruning	2	0	0%
	path pruning	2	0	0%
UDP/20002	node pruning	3	0	0%
	path pruning	3	0	0%
UDP	node pruning	10	3	30%
	path pruning	13	6	46%

Table 6.4: Number of total & *hidden* Flow IDs per protocol and per pruning heuristic, for non-*boot* events

path pruning generally extracts more *hidden* protocols than *node pruning*.

A protocol which notably only appears in *hidden* Flow IDs when using *path pruning* is VoIP. It is only used by the Xiaomi camera when streaming its video feed. It seems that, by default, it does not use this protocol, but resorts to it when the default communication pattern fails.

6.5.3 DNS data: domain names & DNS resolvers

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

- **If the servers the device tries contacting are unresponsive, will it contact servers with other domain names?** To assess this question, we count, for each event, the number of unique domain names contacted.
- **If the device's default DNS resolver is unreachable, will the device try other ones?** For this question, we count the number of unique DNS resolvers queried.

To be more precise, we split both of those measurements between the *first-level*³ and the *hidden* ones.

6.5.3.1 Boot events

Figure 6.9 shows, for each device's *boot* event, the computed DNS-related data: number of domain names contacted, and number of DNS resolvers queried, both split between *first-level* and *hidden*.

Servers' domain names We observe that, out of the 11 *boot* events, seven consider contacting at least one alternative domain name if the default one fails.

DNS resolvers Out of the 19 DNS resolvers contacted, only three are *hidden* resolvers. One of them is the LAN gateway (192.168.1.1), which is contacted by the Tuya plug when it tries to resolve the domain name `a3.tuya-eu.com` as a *hidden* Flow ID. It seems that, by default, this device does not need

³In this context, the qualifier *first-level* indicates the occurrences at the signature tree's first depth level; it has no relation with the similarly named, but unrelated, domain names' top-level domain.

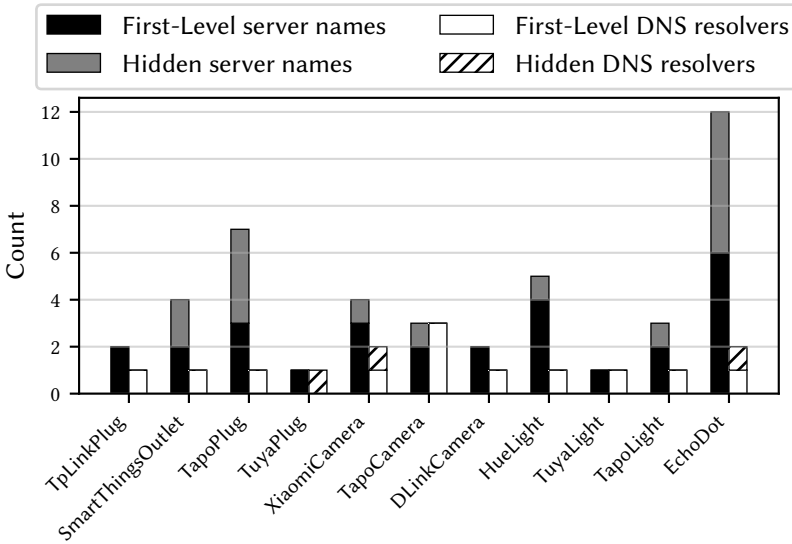


Figure 6.9: Number of unique domain names and DNS resolvers contacted, for each device's *boot* event

to issue any DNS query; this is probably an indication that most addresses it must contact are hard-coded in its firmware, which is a bad practice. When one of them fails, though, it does not fail directly and still uses DNS to resolve the domain name, which provides robustness. The DNS resolver used is the LAN gateway, which is configured as the default DNS resolver for the LAN; it cannot be considered as a *backup* resolver *per se*, but mostly as a default resolver which is only used for *hidden* Flow IDs. Nevertheless, the two others can be considered as real backup resolvers. The Xiaomi camera uses the Chinese public DNS resolver located at the IPv4 address 114.114.114.114, whereas the Amazon Echo Dot uses Google's public resolver with the IPv4 address 8.8.8.8; both are contacted when the LAN gateway is not responsive.

6.5.3.2 Other, non-*boot* events

Figure 6.10 shows the number of *hidden* domain names contacted, under both or either of the pruning heuristics.

The vast majority of the surveyed events, namely 24 out of 27, showcase at least one *hidden* domain name that they alternatively contact when the default one(s) is (are) blocked.

Regarding the DNS resolvers, we found that only one event contacted *hidden* DNS resolvers: the *stream* event, performed by the Xiaomi camera. One

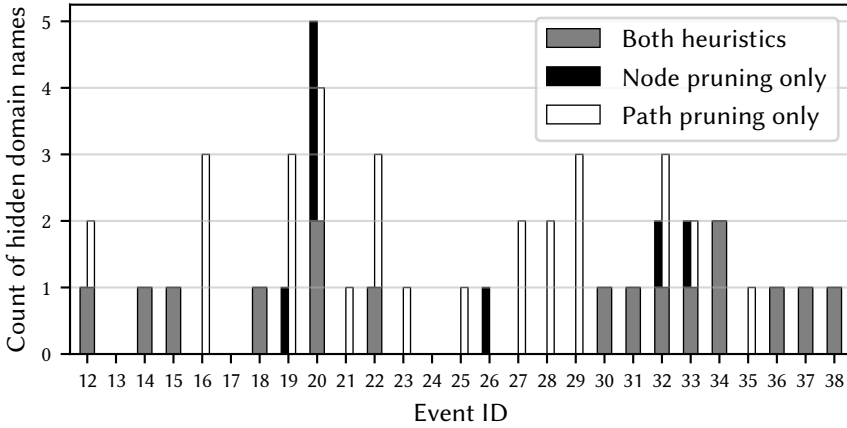


Figure 6.10: Number of *hidden* domain names contacted, for both pruning heuristics, for non-*boot* events

was again the LAN gateway (192.168.1.1), and the other was the aforementioned Chinese public resolver 114.114.114.114.

6.5.3.3 Discussion

Focusing on DNS-related data allows us to sharpen our conclusions regarding the robustness of SH devices. In terms of domain names, most events (31 out of 38, more than 80%) provide a certain degree of robustness, by having a list of alternative domain names that they contact to ensure an event’s success. For three *boot* events and 11 other events, this list of alternative domains even contains more than one, further strengthening the event’s robustness.

However, the observation regarding DNS resolver tones this conclusion down a bit. The fact that only two devices consider trying a backup resolver means that, in general, SH devices still lack robustness in this regard. Indeed, every device should try contacting a backup resolver if the primary one fails, as this has been made easy thanks to publicly available resolvers such as the two aforementioned ones, or Cloudflare’s well-known 1.1.1.1.

6.6 Assessment of events’ robustness

As hinted in the introduction (Section 6.1), SH devices must provide robust interactions to live up to their expectation of easing their user’s household lives. Indeed, if a specific communication path is broken, a device should provide an alternative one for the interaction to still succeed. Yet, manufacturers tend to prefer adding as much functionality as possible on the devices’

modest hardware over consolidating their base behavior. One of the main incentives behind this work is, therefore, evaluating said robustness of SH devices. More precisely, a device can be considered robust if it provides alternative communication patterns designed to ensure the success of a given interaction. The more patterns relative to one interaction, the more robust the device is.

This analysis of the devices' robustness will be conducted in three steps. We start, in Section 6.6.1, by simply counting, for each covered interaction, the number of *hidden* Flow IDs, which are the ones occurring only when a parent one is blocked. They embody the aforementioned alternative communication patterns, thus are a good preliminary metric of an interaction's robustness.

The incentive behind those *hidden* patterns is to avert the interaction's failure; the more patterns an interaction provides to avoid failing, the more robust the interaction is. In Section 6.6.2, we provide additional insight on event robustness by measuring the number of different Flow IDs which must be blocked before the event fails.

Finally, we further analyze how blocking certain Flow IDs might make an event fail. We provide a full-fledged analysis of the probability that an event fails if one or more Flow IDs are blocked. For easier reading, we split our analysis into three subsections covering its three subsequent stages:

- Section 6.6.3: Computing the weight of each individual Flow ID towards an event failure.
- Section 6.6.4: Failure probability when blocking all traffic pertaining to a given protocol.
- Section 6.6.5: Failure probability when blocking a generic set of Flow IDs.

6.6.1 Number of *hidden* Flow IDs

We start the evaluation of the device's communication robustness by counting the number of *hidden* Flow IDs for each covered interaction. We employ a method analogous to that used to count all unique Flow IDs, described in Section 6.5.1.1, but considering only *hidden* ones. Those are the ones which first appear at a depth of two or deeper in the event signature tree, *i.e.* only if their parent had been blocked. They can be analogously defined as the Flow IDs for which there is no corresponding tree node having the root as parent. We define such set of *hidden* Flow IDs as H :

$$H := \{\ell(n) \mid \nexists n \in N : (r, n) \in E\} \quad (6.4)$$

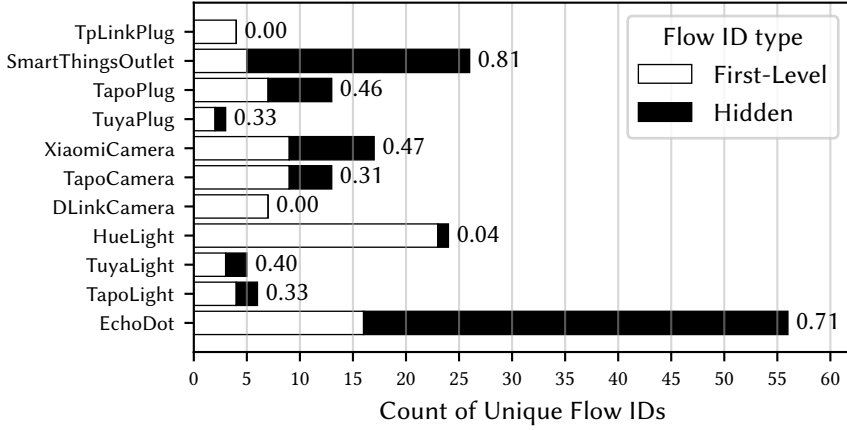


Figure 6.11: Number of unique Flow IDs, for each *boot* event. The hidden Flow ID ratio is displayed next to each bar.

where N is the set of nodes, r is the root node, E is the set of edges, and ℓ the function which assigns a Flow ID to each node. We then simply compute the cardinality of H , namely $|H|$.

By itself, the absolute number of *hidden* Flow IDs does not give much insight. As a *hidden* Flow ID is expected to be a backup communication pattern when a *first-level* one fails, a more interesting metric is the ratio of the number of *hidden* Flow IDs over the total number of unique ones. This ratio is defined as:

$$r := \frac{|H|}{|U|} \quad (6.5)$$

with $|H|$ based on Equation 6.4, and $|U|$ based on Equation 6.3. We will refer to it as the *hidden Flow ID ratio* in the following.

We articulate our measurements in the same way as described in Section 6.5: we separate the data for *boot* events, for which we only applied *node pruning*, from those of all other events, for which both pruning heuristics were applied.

6.6.1.1 Boot events

Figure 6.11 shows the number of unique Flow IDs, computed over each device's *boot* event. They are split between the *first-level* Flow IDs, and the *hidden* ones.

For nine devices out of 11, the framework extracted at least one Flow ID: most of them provide, to a certain extent, robustness in cases where some traffic is not successful. Three of them, namely the SmartThings Outlet, the Xiaomi camera, and the Amazon Echo Dot, even exhibit a higher number of

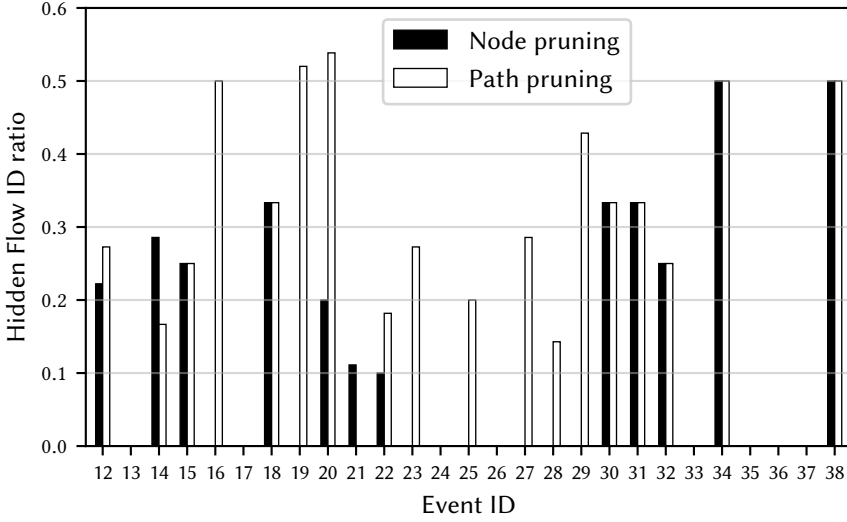


Figure 6.12: Ratio of *hidden* Flow IDs over all unique Flow IDs, for both pruning heuristics

hidden Flow IDs than of *first-level* ones, which demonstrates the high robustness of those devices.

6.6.1.2 Other, non-boot events

The ratio of the number of *hidden* Flow IDs over the total number of unique ones, as defined by Equation 6.5, is displayed on Figure 6.12.

We observe that most of the studied events, *i.e.* 19 out of 27, exhibit at least one *hidden* Flow ID, *i.e.* have a ratio over zero. Moreover, and as already observed in the previous Section 6.5.1, the *path pruning* heuristic is generally more exhaustive than the *node pruning* one, by being able to extract proportionally as many or more *hidden* Flow IDs.

6.6.1.3 Grouped per category

To gain further insight into the distribution of *hidden* Flow IDs for events other than *boot*, we group the previously obtained ratios along two different categories, namely:

- the device category (power plug, light bulb, camera, or speaker);
- the companion app used to control the device.

Here, we consider only the values obtained with the *path pruning* heuristic, as we concluded, in Section 6.5.1, that it is the more exhaustive heuristic.

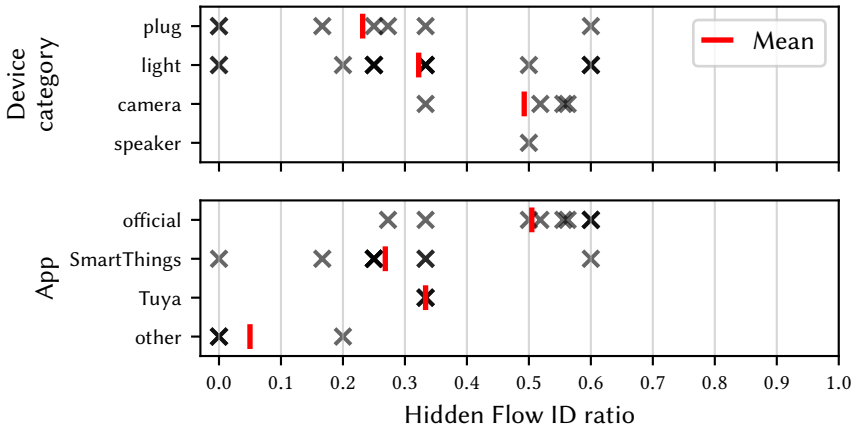


Figure 6.13: Ratio of *hidden* Flow IDs over all unique Flow IDs, both discovered by *path pruning*, grouped per device category and companion app.

Each marker represents the value for one event.

The values are shown on Figure 6.13. If a group comprises more than one value, the value averaged over the whole group is displayed as well. In the following, we break down the values for both of the aforementioned groups.

Device category The values for the power plugs and light bulbs are quite similar, albeit a bit higher for the latter. Both these categories have a similar operation: short actions triggered by the user, embodied by sporadic bursts of traffic. Cameras, however, exhibit an intrinsic difference in operation, when streaming their video feed: they require a steady communication for the whole duration of the stream. Their higher *hidden* Flow ID ratio shows that the manufacturers have strengthened the camera’s communications by providing more various possible patterns to achieve the streaming flawlessly, and enable continuous monitoring of the home, even when some traffic patterns fail.

The value for the smart speaker, namely the Amazon Echo Dot, which only has one event (a voice-controlled command) is exactly 0.5. This means exactly half of all its unique Flow IDs are *hidden* ones. Manual inspection showed it actually has only two unique Flow IDs, and therefore one *first-level* one, and one *hidden* one, used as a backup pattern for the former. This is completely flipped compared to the number of total and *hidden* Flow IDs for its *boot* event, shown on Figure 6.11, which was the highest of all devices. This device seems to perform most of its tasks, e.g. device configuration, resolving domain names, etc. at start-up and/or periodically, but not when a

user interaction is performed. On the one hand, this reduces the traffic tied to each event execution, which incidentally speeds them up. On the other hand, this increases the volume of background traffic issued by the device, including traffic which might actually be unnecessary for the device's actual operation, if the related interactions are never performed. Anyway, for this device, the only *first-level* is still "protected" by a *hidden* one, so it is actually quite robust.

Companion app Unsurprisingly, a device's official app is, on average, the most robust app one can use to control the device. Such apps are generally specifically designed for the devices they are intended to control, and can therefore provide various tailored means of communication. This does not stand for the Tuya-branded devices, though. Tuya does not provide devices directly, but a cloud-based platform to control fleets of IoT devices, as well as the corresponding device firmware. While the Tuya app is the official app to control such devices, the fact that it can be used by tons of devices from different vendors prevents it from being tailored to specific ones. Moreover, the Tuya app does not provide LAN-based device control, and forces the communications to go through their cloud. This reduces by design the robustness of Tuya devices, as, when the cloud endpoints cannot be reached, the device cannot function properly. Besides, we observe that the values for the non-*boot* category are very low.

Contrarily to official apps, third-party apps do not generally provide interactions as tailored to the device as its official companion app. Still, the numbers for SmartThings reach relatively high values, thanks to the substantial effort put in its development as a very popular home automation platform. However, other, less popular third-party apps exhibit very low values. This suggests that, regarding the interactions' robustness, a user should prefer using trusted and official apps, even if a device's API is theoretically open.

We illustrate the effect of the choice of the companion app with the Philips Hue lamp device and its *toggle* interaction, which we have triggered by the official app (Philips Hue), the SmartThings app, and another third-party app, *Hue Essentials*. The highest number of *hidden* Flow IDs is showcased by the official app (12) followed by SmartThings (9), whilst HueEssentials has the lowest value (4).

6.6.1.4 Discussion

Considering both the *boot* and the other events, we analyzed 38 events in total. Out of those, 28 provide at least one *hidden* Flow ID, which accounts for nearly 75% of our corpus. The importance of this discovery is twofold. First, it gives encouraging insight on device robustness: it shows that most

devices provide backup communication patterns to compensate a potential failure of the default ones. Second, it demonstrates that the network behavior of SH devices, even the seemingly simplest ones, is more complex than it appears at a first glance. Even if the *hidden* patterns do not appear by default, they are nonetheless an integral part of the device's behavior, and should be considered as such in models and profiles of the device, e.g. for an allow-list firewall configuration.

6.6.2 Number of Flow IDs blocked until failure

By design, our framework continues processing the current branch of the signature tree until one of the two following conditions are met:

- The pruning heuristic stops the processing.
- The interaction fails.

To gain more insight about an event's robustness, we evaluate the number of Flow IDs which must be blocked before the interaction fails altogether. The higher the number, the higher the event's robustness, as it means that the event is still functional even when part of its traffic is blocked. The main practical reason of a high value is thanks to several alternative communication patterns able to perform the action, although it is also possible that the blocked traffic is actually not required for the event to function. In practice, this number is computed by taking the length of the path from the tree root towards a node which represent an event failure; therefore, we dub it the *path length*.

For a same event, different tree branches might showcase a different path length before the interaction's failure. We compute this metric over all branches which were stopped due to a failure. The events for which the framework cannot verify the successful execution are not considered for this metric, namely all *boot* events, as well as the Amazon Echo Dot's voice recognition event. We decided to compute the metric over the event signature trees resulting from the *path pruning* heuristic, as those provide a more thorough and exhaustive exploration of all the possible tree nodes. The results are shown on Figure 6.14.

Every path length value above one means that the event still succeeded when a Flow ID was blocked. Only 16 out of the 37 events, i.e. 43%, display event failures when blocking only one Flow ID. Even for those, direct failures are not the most likely, as most failures occur after a path of length longer than one. The mean value for the path length before a failure is 4.03, which means that, on average, four Flow IDs must be blocked before the event fails.

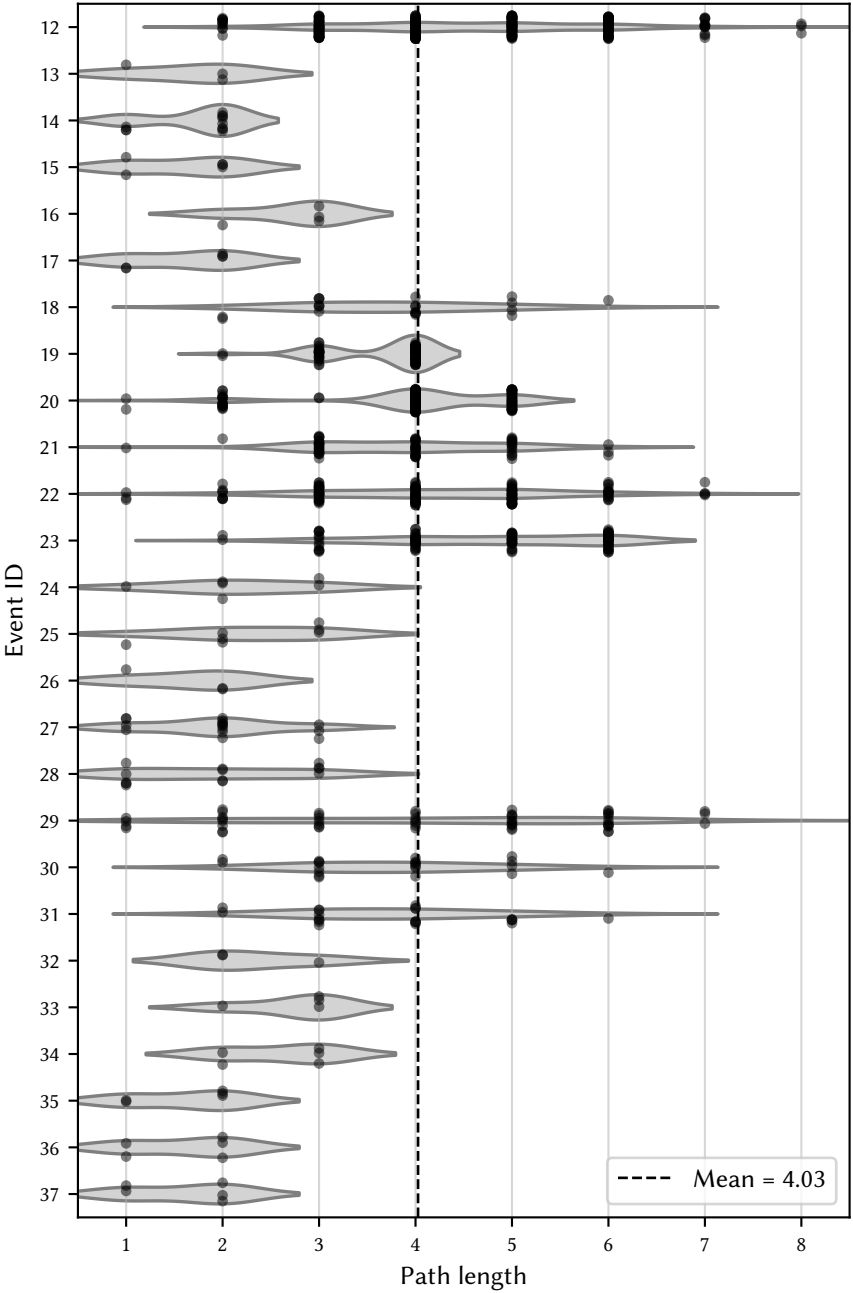


Figure 6.14: Path lengths until event failure

This result is very encouraging regarding the robustness of SH device's communications, and shows that manufacturers did not neglect it while designing SH devices.

6.6.3 Failure causal attribution to Flow IDs

Our objective is to understand to which extent each of an event's Flow IDs, when it is blocked, plays a role in the failure of said event. Here, we derive a formula which computes, for any Flow ID, the probability that the event's execution will fail when it is blocked. This probability is dubbed the Flow ID's *causal strength*.

With F the set of all possible Flow IDs, we define $F_e \subseteq F$, the set of Flow IDs blocked during execution e of a given event. Additionally, we define y_e as the outcome of execution e , encoded as a binary value (0 or 1):

$$y_e := \begin{cases} 1 & \text{failure} \\ 0 & \text{success} \end{cases}$$

Each Flow IDs $f_i \in F_e$ provides a *causal strength* p_i , which is defined as the probability that a failure occurs when Flow ID f_i is blocked. We assume the impact of each individual Flow ID is independent; this will be discussed in Section 7.2. Based on this assumption, we model the failure probability, considering a set of blocked Flow IDs F_e , denoted as P_e , as a Noisy-OR model [375]:

$$P_e := P(y_e = 1 \mid F_e) = 1 - \prod_{f_i \in F_e} (1 - p_i) \quad (6.6)$$

We consider a set of m event executions, for which we know the outcomes, either $y_e = 1$ or $y_e = 0$. Each execution has a probability of failure P_e , depending on the set of Flow IDs F_e blocked during this execution. Based on this knowledge, we can infer the causal strengths p_i by leveraging Maximum Likelihood Estimation (MLE) [376] over the log-likelihood of the causal strengths, defined as:

$$\log L(p_1, \dots, p_n) = \sum_{e=1}^m (y_e \log P_e + (1 - y_e) \log(1 - P_e))$$

The causal strengths p_i are inferred by minimizing the log-likelihood, based on their values.

6.6.4 Failure probability per protocol

Based on the causal strengths p_i derived in Section 6.6.3, we define P_P as the probability that an event execution k fails when blocking all Flow IDs pertaining to a given protocol P . Let $F_P := \{f \in F \mid f \text{ pertains to } P\}$. Therefore,

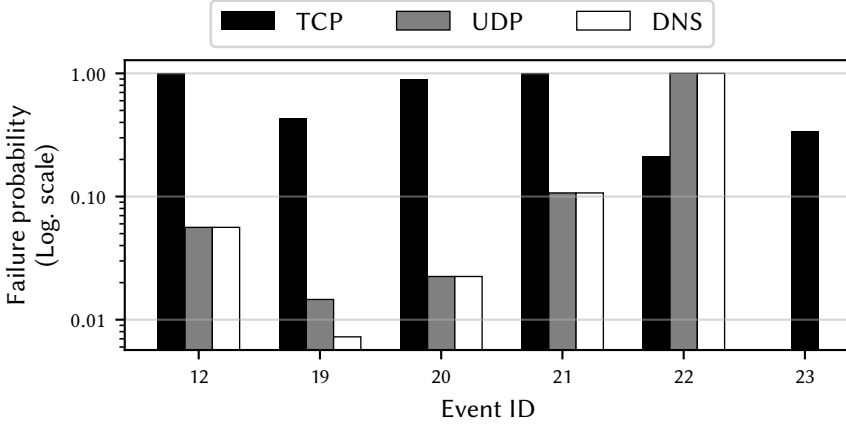


Figure 6.15: Event failure probability per blocked protocol. The events not shown have a probability of 1 for TCP, and 0 for all other protocols.

we define P_P using Equation 6.6:

$$P_P := P(y_P = 1 \mid F_P) = 1 - \prod_{f_i \in F_P} (1 - p_i)$$

For each event, we compute P_P for each involved protocol. The first striking observation is that P_{TCP} is 1, for nearly all surveyed events. This means that, in most cases, blocking TCP is guaranteed to hinder the event. This is expected, as the most commonly used protocol is HTTPS, which is based on TCP. Blocking TCP altogether will not only prevent HTTPS, but also HTTP, or even plain TCP traffic, the latter being commonly observed in the direct communication between a device and its companion app.

On Figure 6.15, we display the values for P_{TCP} , P_{UDP} and P_{DNS} for the remaining events having P_{TCP} under 1, or one of the others over 0. Note that, in our measurements, the application-layer protocol DNS was systematically based on UDP at the transport layer; therefore, the value for UDP is always at least equal to that of DNS.

Blocking UDP has a way less significant effect than TCP, in that it only impedes five of the surveyed events, and only one with a guaranteed failure. Looking at the values for DNS, which are included in the UDP values, we conclude that most of the failures occurring when blocking UDP are actually related to DNS. Blocking DNS traffic means the devices are unable to resolve the IP address corresponding to the domain names they must communicate with, and are therefore unable to contact them. Nevertheless, most events still work correctly even when DNS does not; either because they resolved and stored the necessary domain names upon booting, or because they use hard-

coded IP addresses. The latter strategy is however considered a bad practice, even if, in this very specific case, it helps to preserve functionality.

Only a very small fraction of the UDP traffic for which the blocking can impede the event is not DNS traffic. It is related to one event, namely the *stream* event of the Xiaomi camera. The devices in our corpus which use UDP the most extensively are the three cameras, when they are streaming their live feed. All three have similar strategies to perform the streaming: they first contact the phone directly through UDP in the LAN; if this fails, they contact an intermediary remote server, again through UDP; if both fail, they contact a remote server (not necessarily the same) through HTTPS. Even if the latter is less suited for real-time streaming, the fact that it is used as backup protocol when the UDP fails improves the reliability of the cameras. For the two other cameras, the probability that streaming is prevented by blocking plain UDP (not DNS) is 0, as HTTPS will always take over. However, for the Xiaomi camera, there is a specific case where blocking plain UDP only is sufficient to disrupt the streaming. This camera can contact two different remote servers as streaming intermediaries. If both are unreachable through UDP, the streaming will fail systematically. For this camera, it seems that at least one of the intermediary remote servers must be reachable via UDP to achieve the streaming.

6.6.5 Failure probability when blocking a set of Flow IDs

We wrap up the assessment of the events' robustness by computing a generic probability of failure based on the number of Flow IDs blocked, regardless of their nature.

Consider an event signature tree T resulting from an event execution e . We define the function $p(k)$ as the probability of failure when blocking a set of any k Flow IDs. Its derivation is not trivial, and is explained in the following.

6.6.5.1 Derivation of the formula

Let $F_{k,i}$ be a given set of Flow IDs of cardinality k . The failure probability when blocking all the Flow IDs $f_i \in F_{k,i}$ is given by Equation 6.6, by replacing F_e with $F_{k,i}$. Let F_k be the set of all sets of Flow IDs of cardinality k :

$$F_k := \{F_{k,i} \mid i \in [1, \infty), |F_{k,i}| = k\}$$

We assume all $F_{k,i}$ have an identical probability of being picked at random, denoted P_k . This probability is therefore simply the inverse of the cardinality of F_k :

$$P_k := \frac{1}{|F_k|} \quad (6.7)$$

With U being the set of unique Flow IDs in T , as defined by Equation 6.3, we note $n = |U|$ as the total number of unique Flow IDs in T . The cardinality of F_k ($|F_k|$) is the number of different sets of Flow IDs of size k . Considering there are, in total, n different Flow IDs, $|F_k|$ is computed as the number of combinations of k elements from a set of size n , i.e. the binomial coefficient $\binom{n}{k}$:

$$|F_k| = \binom{n}{k} = \frac{n!}{k!(n-k)!} \quad (6.8)$$

Plugging the value from Equation 6.8 into Equation 6.7, P_k can thus be computed as follows:

$$P_k = \frac{k!(n-k)!}{n!} \quad (6.9)$$

The joint probability for two dependent events A and B is given by Equation 6.10:

$$P(A \wedge B) = P(A) \cdot P(B | A) \quad (6.10)$$

We compute the probability that a specific set $F_{k,i}$ of k Flow IDs is picked to be blocked **and** a failure occurs due to those Flow IDs being blocked by inserting Equations 6.9 and 6.6 into Equation 6.10:

$$\begin{aligned} P(F_{k,i} \wedge y_{k,i} = 1) &= P(F_{k,i}) \cdot P(y_{k,i} = 1 | F_{k,i}) \\ P(F_{k,i} \wedge y_{k,i} = 1) &= P_k \cdot \left(1 - \prod_{j=1}^k (1 - p_j) \right) \\ P(F_{k,i} \wedge y_{k,i} = 1) &= \frac{k!(n-k)!}{n!} \cdot \left(1 - \prod_{j=1}^k (1 - p_j) \right) \end{aligned} \quad (6.11)$$

Any set of k Flow IDs can independently cause a failure when blocked. Finally, to compute $p(k)$, i.e. the probability of failure when a random set of k Flow IDs is picked and blocked, we sum Equation 6.11 over all different sets of k Flow IDs, i.e. over all elements comprising F_k . Recall that $|F_k| = \binom{n}{k}$.

$$\begin{aligned} p(k) &= \sum_{i=1}^{\binom{n}{k}} \left(\frac{k!(n-k)!}{n!} \cdot \left(1 - \prod_{j=1}^k (1 - p_j) \right) \right) \\ p(k) &= \frac{k!(n-k)!}{n!} \cdot \sum_{i=1}^{\binom{n}{k}} \left(1 - \prod_{j=1}^k (1 - p_j) \right) \end{aligned} \quad (6.12)$$

6.6.5.2 Computation

After having derived the formula defining P_k , we compute its value over all the interactions surveyed by our study. The results are shown on Figure 6.16, split per device category:

- Figure 6.16a: Power plugs
- Figure 6.16b: Light bulbs
- Figure 6.16c: Cameras

The most frequent shape is a curve with a decreasing rate of increase as the value approaches a probability of 1. The remaining ones seem to adopt a linear increase from 0 to 1. The increase rate is quite different between devices, though. We observe that the cameras' is slower than that of the two other device categories. As already pinpointed in Section 6.6.1.3, this can be explained by a difference in operation between, on the one hand, plugs and light bulbs, and on the other, cameras. The former rely on sporadic bursts of traffic; if an event execution fails, it can be easily retried shortly after. Cameras, however, require constant communication when streaming; if stopped, a lot of important information might be lost. Moreover, they are usually tied with more critical functions, such as monitoring a home against potential intrusions. For these reasons, it seems that camera manufacturers put additional effort in ensuring their device's robustness.

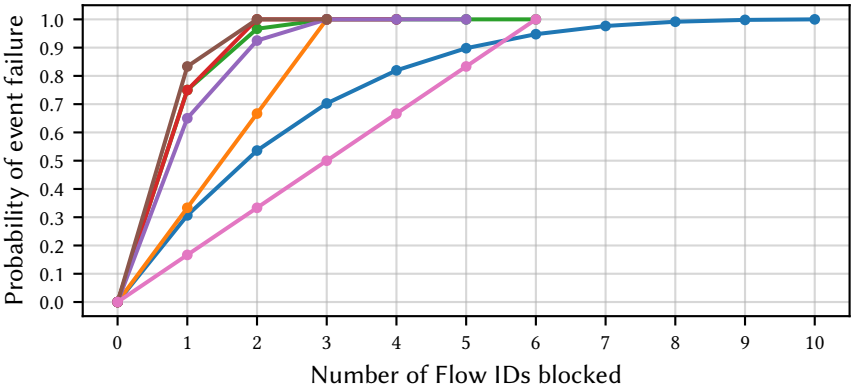
A visible outlier in the light bulbs category is the Philips Hue's *toggle* event. We can gain insight on the reason by taking a look at Figure 6.14 again; this event's ID is 23. We observe that the path length before this event fails is generally quite high: its minimum is two, with only two paths having this length, and a lot of paths exhibit values above two. It means that, in general, this event is very robust, and its failure requires the blocking of many Flow IDs.

6.6.6 Takeaway: robustness guidelines

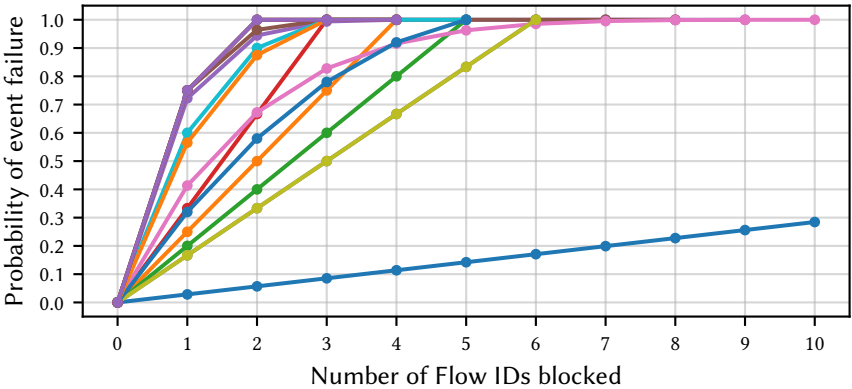
To summarize the robustness assessment of SH devices, we provide guidelines a device should follow to provide robust communication.

6.6.6.1 Various *hidden* Flow IDs

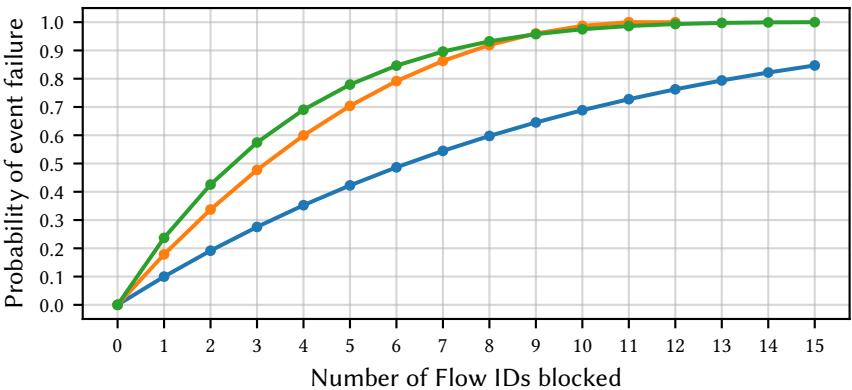
The prime criterion is for a device to provide a wide range of *hidden* Flow IDs, which will act as backup communication patterns whenever the default ones cannot succeed. If a device does not provide such patterns, it will stop functioning as soon as the default ones fail. To be as effective as possible, such backup patterns should exhibit different network characteristics than the one they are meant to replace, *e.g.* use a different protocol, or contacting another domain name. This ensures that, if the initial characteristic is blocked altogether, the backup pattern can circumvent the blocking. For instance, a default communication pattern using TCP can be replaced by one using UDP instead.



(a) Device category: Plugs



(b) Device category: Lights



(c) Device category: Cameras

Figure 6.16: Probability of event failure when blocking a random set of Flow IDs

6.6.6.2 Various DNS resolvers

Before contacting a server via its domain name, said domain name must be resolved by querying a DNS resolver. The latter can be the LAN gateway, a specific resolver in the LAN, or even remote resolvers, such as public ones from Cloudflare (1.1.1.1) or Google (8.8.8.8). The default DNS resolver to use can be specified by the DHCP server, included in the DHCP Offer message, along with the proposed IP address for the device when it enters the network. However, if only one resolver is configured, it becomes a single point of failure. Indeed, if the communication with this resolver is broken, the device would not be able to resolve any domain name, and would be potentially blocked from contacting any remote server. Therefore, manufacturers can easily increase robustness by configuring backup DNS resolvers to query if the default one is not responsive. Usually, public resolvers are used as such, as one can assume those will be virtually always available.

6.6.6.3 Using domain names instead of direct IP addresses

We wrap up with a more generic and very well-known criterion, though it is still not fully applied in the IoT context. Usually, when contacting servers located on the Internet, devices address them using domain names, which act as a proxy for their IP address, the latter being actually used to route the traffic. Most vendors' servers are hosted by worldwide cloud providers: in the traffic we generated throughout this study, they were all hosted by Amazon Web Services (AWS). The servers' IP addresses are therefore very volatile, changing along the cloud provider's orchestration. Using domain names ensures a server is reachable regardless of its actual IP address, whereas using the latter directly will likely end up in a broken communication whenever the cloud provider changes its data center's structure. Besides, a single domain name can be resolved to diverging IP addresses, to provide load balancing, or depending on the geolocation of the querying device.

Nevertheless, contacting servers via their domain names introduces an extra step in the communication chain, namely resolving the domain name. As explained above, this step's robustness should also be considered. It also requires registering a domain name for the server in the first place. To alleviate this process and streamline the communication, one can prefer using direct IP addresses. This rationale should only be followed when the server is hosted by the vendor directly, and is guaranteed to stay constant. It would also be helpful in the unlikely case where all DNS traffic would fail.

Devices	Apps	PingPong [22]	BehavIoT [23]	MUDgee [346]
TP-Link HS110	Kasa Smart	toggle	toggle	device
Hue lamp	Philips Hue	toggle	✗	device
SmartThings plug	SmartThings	toggle	boot	device
D-Link camera	mydlink	✗	stream	✗

Table 6.5: Related works' device and interaction coverage.

6.7 Comparison of results

In this section, we compare our results with those produced by related works, first regarding the total number of unique Flow IDs, then focusing on domain names.

6.7.1 Number of Flow IDs extracted

We compare the profiles generated by our framework with those generated by methods proposed in related works, namely *PingPong* [23], *BehavIoT* [346], and *MUDgee* [22], for the devices present in both their testbeds and ours. More precisely, we count the number of Flow IDs extracted in all cases, as a measure of the approach's exhaustiveness. As previously, for our work, we consider two different values, based on the pruning heuristic which was used to extract the Flow IDs for events other than *boot*; for the latter, only *node pruning* was applied. The covered related works must be separated based on their level of abstraction. On the one hand, *PingPong* and *BehavIoT* profile specific interactions of a device, what will be dubbed as *interaction-level* profiling. On the other hand, *MUDgee* profiles the device as a whole, indivisible unit, which we will refer to as *device-level* profiling. By design, all three works do not explore the event signature tree deeper than the first level; we therefore expect to have steadily discovered more Flow IDs through our approach. Table 6.5 summarizes the devices and interactions covered by each work, as well as the smartphone apps which produced the subset of our data used for our side in the comparison.

6.7.1.1 Interaction-level profiling

Regarding interaction-level profiling, Figure 6.17 compares the number of unique Flow IDs discovered by *PingPong*, *BehavIoT*, and us. For our work, we display the number obtained with both the *node pruning* and the *path pruning* heuristics. The data for *PingPong* were taken from their reported results, whereas *BehavIoT*'s results were reproduced on our side. Our approach is, as expected, able to extract more Flow IDs than *PingPong* and *BehavIoT*,

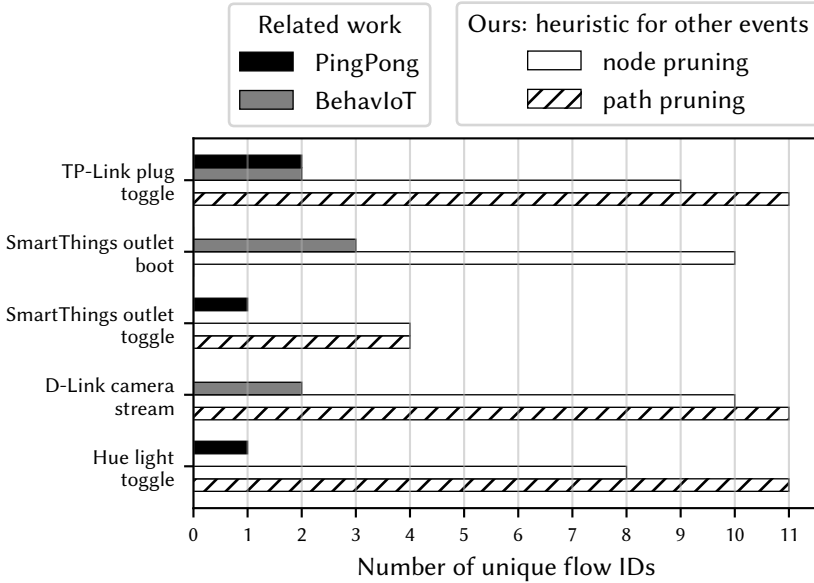


Figure 6.17: Comparison of discovered unique Flow IDs between two related works and our two pruning heuristics, at the interaction level.

thanks to a deeper tree inspection, and also a more thorough extraction of the communication patterns, even at the first level. Moreover, as already shown, our *path pruning* heuristic allows discovering more Flow IDs than the *node pruning* one.

6.7.1.2 Device-level profiling

Figure 6.18 shows the number of unique Flow IDs discovered with a device-level strategy. To compute our values, for a device, we aggregate the unique Flow IDs over all interactions of the device. The data for *MUDgee* were taken from their reported results. For two of the studied devices, namely the SmartThings Outlet and the Hue light, our approach is again able to extract a significantly higher number of Flow IDs than *MUDgee*. However, for the TP-Link HS110 plug, we find the same number of Flow IDs when using *path pruning*, i.e. 14, and only 12 when using *node pruning*. A manual investigation shows that the nature of the extracted Flow IDs is different: 10 out of 14 *MUDgee* flows are NTP traffic towards various servers, whereas our solution did not extract any NTP patterns. This traffic might be part of periodic patterns, which are not covered by our approach; as explained in Section 6.3.3, our methodology excludes traffic that does not appear in every iteration of a tested interaction from the profile building process. On the other hand, we

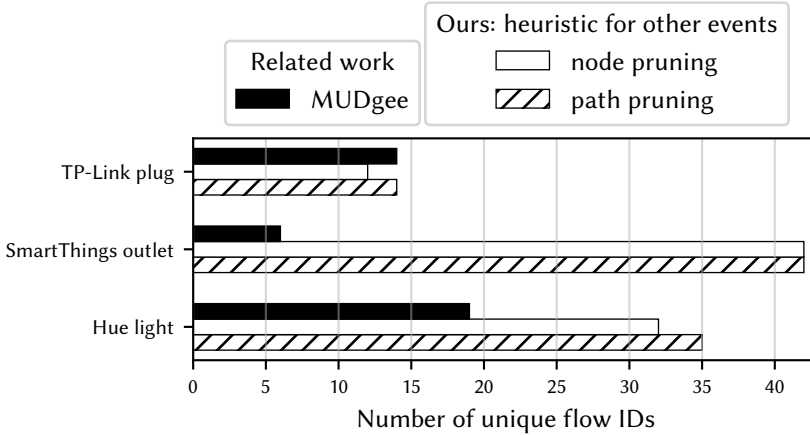


Figure 6.18: Comparison of discovered unique Flow IDs between MUDgee and our two pruning heuristics, at the device level.

extract 10 non-NTP traffic patterns which were not detected by *MUDgee*, including communication with TP-Link servers which are probably alternate communication paths that provide a backup path if the default one fails.

6.7.2 Domain names extracted

Additionally, we compare the number of domains names discovered by our approach with those reported by Mandalari *et al.* [314]. The latter focuses on extracting the hosts from SH communication patterns. Table 6.6 shows the total number of unique domain names discovered by our approach and by theirs for the devices and interactions covered in both works. We observe that our multi-level approach is able to discover more domain names. However, it should be noted that changes in the firmware of the devices since the publication of the cited article [314] could influence the results.

A direct, sound comparison of the domain names beyond the mere numbers is unfortunately not possible. This is because Mandalari *et al.* were primarily interested in finding “non-essential” hosts, *i.e.* hosts that can be blocked, for example for privacy reasons, without negatively impacting the essential functionality of a SH device. Consequently, while the total numbers are known, they only published the domain names of non-essential hosts, whereas we do not differ between essential and non-essential ones.

Nevertheless, we still compare the domain names discovered by our approach and theirs; Figure 6.19 shows said domain names, split per device. The differences between the two works can be explained by the reasons already mentioned above. For example, the domain name `ecdinterface.philips`.

Device Interaction	Ours	Mandalari <i>et al.</i> [314]
TP-Link HS110 boot & toggle	6	4
SmartThings Outlet boot	4	3
Philips Hue lamp boot	7	4

Table 6.6: Number of domain names extracted by Mandalari *et al.* [314] and our study.

com does not exist anymore (in 2025), and the domain name diagnostics.meethue.com could be part of a periodic or sporadic pattern filtered out by our approach. This can also explain the empty intersections between the two works for two of the tested devices.

6.8 Related work

The emergence of IoT devices in numerous settings in our daily lives has motivated researchers, similarly to us, to characterize the communication patterns such devices exhibit. The IoT being a widely encompassing term, there might be as many different patterns as there are devices, which entails a variety of research projects on the topic. In this section, we will summarize such related works, and position them compared to our work. As a preliminary note, none of the works mentioned here actively attempted to discover hidden communication patterns. The fact that we consider those as an integral part of a device’s traffic is a noteworthy added value.

6.8.1 Smart Home traffic signature extraction

The popularity of the IoT as a whole, which includes purpose-driven devices issuing simple and predictable traffic patterns, has prompted the IETF to provide a format to express such intended patterns for a device, culminating in the standardization of MUD. The latter was extensively described in Section 5.2.2. Once the format is specified, the next logical step is to design tools able to automatically generate profiles complying with the format. Hamza *et al.* [364] proposed a solution to generate MUD profiles from network traces of IoT devices’ traffic. The generated profiles, however, suffer from the shortcomings inherent to the MUD standard, highlighted in Section 5.2.3, *e.g.* the lack of support for protocols other than IP (v4 & v6) at layer 3, and TCP, UDP and ICMP at layer 4.

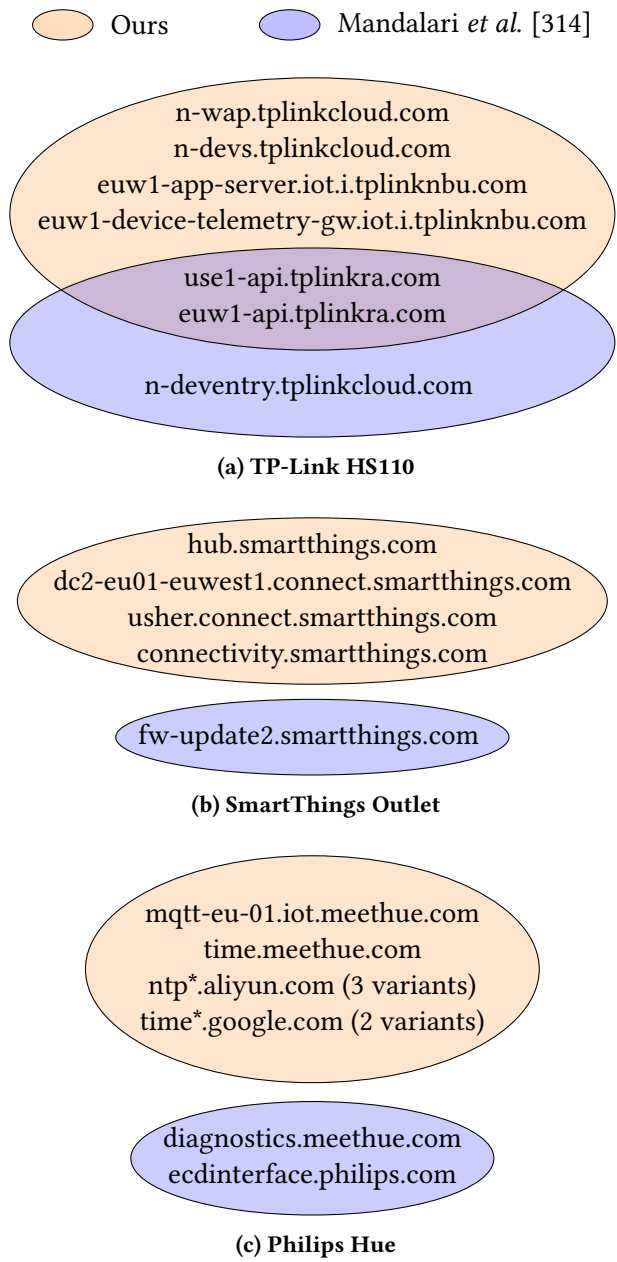


Figure 6.19: Venn diagrams showing the domain names extracted by Mandalari *et al.* [314] and our study.
Note that the domain names actually named by Mandalari *et al.* are only non-essential hosts, as defined in their study.

Our workflow to extract signatures from SH network traffic can be summarized in the following steps:

1. Trigger the event under test
2. Capture the associated traffic

Those two steps are repeated n times in order to obtain a statistically meaningful baseline. Afterwards, the signature extraction algorithm is applied on the set of n traffic captures. This workflow was followed by numerous related works.

Packet-level signatures of Smart Home events

With *PingPong* [23], Trimanada *et al.* extract traffic signatures at a finer level of granularity than our work. They leverage only packet-level metadata, namely the packet length and direction. On the contrary, we considered flow-level protocol fields, *i.e.* hosts, protocol, port numbers, etc. The *IoTA* tool by Duan *et al.* [26] followed a similar approach, and consider traffic signatures only containing packet length and direction as well. If different legitimate packet sequences exhibit a common part, that part will be merged, and shared among the sequences. Therefore, a signature takes the form of a directed graph, where each node represents a packet, and some nodes introduce a branching among several children, those being the possible sequels of a common starting sequence. For their system *D-interact*, Sun *et al.* [27] again produce traffic signatures containing only the packet length and direction. The packet sequence designated as the signature among the n captured ones is the one for which the sum of *Levenshtein distances* [377] with all other sequences is minimal.

The main difference between those and our work is the features considered to build the signature. Theirs are composed of the metadata of individual packets, namely packet length and direction, whereas we aggregate packets based on protocol fields, forming what we dubbed a *Flow ID*. We argue that statistical features tied to single packets might be imprecise to accurately characterize communication patterns, as they are constant only when the network stays perfectly stable. For example, if a TCP packet is lost during transmission, it will be retransmitted, leading to inconsistencies in the measured number of packets. On the contrary, our research aims to extract signatures taking into account potential network instabilities.

Wan *et al.* addressed this issue with their work *IoTAthena* [367]. This project has a similar objective as the three aforementioned ones, that is extracting packet-level signatures for SH events. Nevertheless, besides the packet metadata (length and direction) used in the previous works, they added semantic information from protocol fields at different OSI layers, namely the

hostnames, service port, protocol, and application-layer data. Their approach is therefore a hybrid between ours, which only considers protocol data, and the previous works with only packet metadata.

Others

Mandalari *et al.* [314] centered their analysis on the hosts contacted by SH devices, characterized mainly by their domain names, but falling back to their plain IP address if the domain name was unavailable. Their goal was to identify non-required hosts, *i.e.* hosts which could be blocked without impairing the device's operation, *e.g.* telemetry or advertisement services. While they designed a workflow similar to ours to identify the successful events, they did not further explore the event signature tree.

Hu *et al.* proposed *BehavIoT* [346], a system which models the behavior of a whole SH network, with the intent of using it to detect when the network produces unintended behavior, potentially due to unwanted communication, or even a security breach. Their model is twofold: on the one hand, they model individual devices events; on the other hand, cross-device interactions. Their complete model is generated based on the network traffic produced by the devices, and takes the form of a Probabilistic Finite State Machine (PFSM), representing all the possible transitions between events in the SH network. By covering all individual device events as well as cross-device interactions, their work exhibit a broad scope. Yet, the model of cross-device interactions is *ad hoc* by design, as it can only represent the network in which it was generated, and fails to generalize. Nevertheless, it is still an open question to design a general model of cross-devices interactions, and if this is even practically possible.

6.8.2 Empirical traffic measurements

The works presented above intended to identify the communication patterns issued when SH devices perform a specific action. Other researchers might want to take a step back, to measure and characterize the network traffic related to a Smart Home in its entirety, potentially compiling empirical metrics describing the traffic. Such projects will be presented in the following.

While most researchers tend to focus on the interactions between SH devices and cloud endpoints, Girish *et al.* [17] contrarily observed the communications taking place in the SH LAN. They conducted a comprehensive analysis by exhaustively characterizing the protocols used, and highlighting the security and privacy vulnerabilities of device communications in the LAN.

On the other end of the spectrum, Saidi *et al.* [184] take the point of view of an ISP, which do not have access to singular packets, but to highly aggregated and sampled traffic data. Based on such sparse data, they designed

a methodology able to reliably identify the IoT devices issuing the traffic. As an ISP has a high-level view on traffic coming from multiple smaller networks, being able to identify IoT devices issuing traffic might be largely important, for example to detect a DDoS attack originating from an IoT botnet, as the introductory Mirai botnet [5].

Hu *et al.*, the same authors as *BehavIoT* [346], have published another study, this time analyzing the usage of IPv6 in SH networks [44]. Since its initial specification in 1995 as RFC 1883, IPv6 has been supposed to supersede IPv4. Yet, in practice, IPv4 stays the most widely used network protocol, while IPv6 is still unsupported by numerous services. IPv6 was designed to provide an address space virtually impossible to deplete, by using 128-bits addresses. As the IoT has brought a dramatic amount of new Internet-connected devices in today's networks, one could have thought that IPv6 was fit to address all those IoT devices. However, as IPv6 requires more resources, the manufacturers might be shy in implementing in onto their devices. The study's main result is that, in their test environment, only 8.6% of their devices kept functioning in an IPv6-only network.

For now, we have mentioned works covering devices using the TCP/IP protocol stack. As presented in Section 2.1.3, IoT networks might employ various protocols, depending on the system requirements. In the specific case of Smart Homes, Zigbee and Thread are two popular non-IP protocols, both based on the IEEE 805.15.4 physical layer. Franceschinis *et al.* [378] evaluated the performance of two variants of Zigbee, namely Zigbee Pro and Zigbee IP, when used jointly with a TCP/IP network, a setting mimicking an actual SH environment. Sistu *et al.* [379] covered the Thread protocol, which is more recent than Zigbee, and envisioned to take the spotlight in future Smart Homes as the enabling protocol for Matter [78]. More precisely, they evaluated the fitness of Thread for a typical SH application, namely a smart lighting system, and concluded that Thread is indeed fit for this purpose.

6.8.3 Chaos Engineering

Finally, the concept of chaos engineering [380] is related to our approach. It consists in injecting faults (e.g. network loss or latency) or heavy load (e.g. on CPU, memory, etc.). Examples of tools for chaos engineering include Chaos Monkey [381], Chaos Mesh [382], and Steadybit [383]. While our approach shares some concepts with chaos engineering, its novelty resides in the fact that we are the first to apply it to this extent in the context of smart homes. Moreover, today, no chaos engineering tool provides control over network traffic at a protocol level as precise as us, by blocking traffic based on specific fields. The most precise is Steadybit [383], which allows blocking TCP altogether.

6.9 Conclusion

As the Smart Home, and more generally the IoT, enter every aspect of our daily life, a righteous concern can be raised about the robustness of their operation, that is if they are able to perform the required actions even in non-ideal scenarios. Indeed, we do not want our smart lights to be unable to turn on simply because they do not manage to reach a remote server. In this study, we designed a novel methodology, and implemented it as a framework, with the ultimate goal of assessing the robustness of SH devices. In the context of this work, we approached a device's robustness from the angle of its network traffic. More precisely, a device was considered robust if, in the case where its default traffic cannot reach its destination, said device provides backup communication patterns to alleviate the unsuccessful one.

Concretely, our framework strives to exhaustively extract all communication patterns tied to a given action from a SH device, including those that only occur whenever the default ones fail, which we label as *hidden*. The discovery of such *hidden* patterns has steadily been overlooked by existing works, which only focused on the tip of the traffic iceberg. The outcome of our framework, for a given action, is a comprehensive profile which takes the form of a tree, where each node is a traffic flow associated with the action.

Subsequently, we defined various metrics intended to express the robustness of the surveyed devices. Good news: most devices already provide, to a certain extent, robustness towards network instability. 28 out of the 38 tested events provide at least one *hidden* backup communication pattern, and the average number of such patterns is 4.6 per event, with our most exhaustive heuristic. We pushed the analysis further by modeling the probability that a SH event will fail, based on the traffic flows which are blocked, e.g. by a firewall in the local network. We can envision that, in the future, a similar metric might be used as a marketing argument; indeed, customers would prefer a device which is robust and able to function even in inhospitable network conditions. As a takeaway from our study, we emphasize three guidelines which enhance a device's robustness: providing backup communication patterns; having a list of backup DNS resolvers; contacting servers via their domain name instead of their IP address.

Besides robustness itself, our study highlights that an IoT device might produce traffic diverging from the expected baseline in specific conditions. While this traffic is indeed rare, it is nevertheless an integral part of the device's normal behavior, and should be considered as such. This discovery has implications for all systems relying on a model of a device's normal behavior, e.g. a security system which would be configured to allow traffic complying with a model of the device's expected traffic. It should be ensured that such a model contains all the traffic the device can potentially produce, oth-

erwise it would wrongly consider such traffic as malicious. Our framework can help do so, by creating a more exhaustive model of the device's traffic. Additionally, analogously to Mandalari *et al.* [314], while this was not the focus of our study, the exhaustiveness of the framework also allows uncovering undesirable, or at least unnecessary, hidden patterns, *e.g.* data exfiltration, or alternative communication paths towards foreign servers, which could be taken into account and filtered out.

An early version of the work presented in this chapter was presented at the 2025 International Conference on Network Protocols (ICNP) [3]. A complete version including all the chapter's content is currently in preparation, which we expect to publish in a journal.

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

Discussion & Perspectives | 7

Throughout this thesis, we developed ways to express the network communication patterns issued by SH devices in the form of finite profiles. The underlying objective is for a profile to exhaustively describe all patterns of the related device, such that it forms a complete model of the device's network traffic. In this way, anomaly-based security systems, *e.g.* allow-list firewalls, could leverage the profile to provide accurate decisions over the passing traffic.

In this chapter, we highlight four practical directions for future research to extend the scope of our contributions, covering the following subjects:

- The automatic generation of a SH device's behavioral profile, considering the level of abstraction of its content.
- The assumption, in Section 6.6.3, that each blocked Flow ID contributes independently to the event failure.
- The discovery of traffic patterns variations due to geolocation.
- A shift, from the opaque-box strategy we followed in Chapter 6, *i.e.* considering only the device's input and output (in this case, its traffic patterns), towards a clear-box approach, leveraging information about the device's or its companion app's source code. Indeed, with an opaque-box strategy, ensuring whether all the patterns have been discovered is an undecidable problem.

For each, we identify the potential research perspectives, and, if relevant, add some insight on how to apply them in practice.

The perspectives presented in this chapter are more general, as the ones that are shorter-term and more closely tied to a given contribution have been introduced in the associated earlier chapter.

7.1 Automatic profile generation

In Chapter 5, we designed a format to, hopefully, exhaustively express the communication patterns tied to a given SH device. Once the format is defined, a question which is raised is that of the generation of profiles complying with

it. This question was introduced in Section 5.6.4. In Chapter 6, we designed a framework with the objective of addressing this question. Our framework employs SH devices, and, provided a small initial bootstrap regarding the SH event under scrutiny, automatically extracts an in-depth traffic signature for said event.

The profiles described in Chapter 5 intend to model a SH device as a whole. Their content is traffic features, characterizing either singular packets, or flows, *i.e.* aggregates of several packets sharing given features. In contrast, our methodology derived in Chapter 6 extracts a signature for a specific SH event of a device; in that sense, it is more fine-grained. However, given the exhaustiveness we strive for in the event signature, it would be impossible to characterize each individual packet associated with the event. The signatures are composed of flows which abstract packet features, and, as such, cover a set of such packets.

These key differences illustrate the potential disparity in level of abstraction when profiling SH-related communication. We identify two axes along which the abstraction level can vary:

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

Picking the correct abstraction level always stems from a compromise between precision and complexity. The rule of thumb is that a lower abstraction level enhances the precision of the profiling, as it allows for a fine-grained vision on the communication. However, the cost is often bigger profiles, not only in size but also in complexity, which can be more tedious to work with. Table 7.1 depicts both aforementioned abstraction axes placed orthogonally, and the nine possible combinations of values from both axes.

In the remaining of this section, we will discuss each one of this table's cells, give the pros and cons of the associated approach, and examples of existing works implementing it.

(a) Object: Event; Behavior: Packets

We start with the least abstract cell of the table: the profiling of SH events based on the network packets transmitted during its execution.

Three related works applying this strategy were already discussed in Section 6.8.1: *PingPong* [23], *IoTA* [26], and *D-interact* [27]. Their signatures of

		Object →		
		Event	Device	Network
Behavior ↓	Packets	(a)	(b)	(c)
	Flows	(d)	(e)	(f)
	Events	(g)	(h)	(i)

Table 7.1: Abstraction levels of Smart Home profiling.
The arrows point towards increasing abstraction levels.

SH events all leverage the same features: the length and direction of packets associated with the event. By design, this approach results in fine-grained signatures. Once they have been derived, the signatures can be leveraged to detect the event in new traffic with very high precision. However, the packet metadata used as the signatures' building blocks render the signatures highly volatile. Indeed, if a packet differs only in one byte of length, it does not comply with the signature, even if it actually represents the same event. If an event exhibits even small variations in its traffic, a standalone signature must be derived for all variations; this becomes even more tedious if those variations are unbounded and/or unpredictable. Besides, packet retransmissions might disturb the signature.

Two works following a similar approach have considered this issue. With *IoTathena*, Wan *et al.* [367] produce very similar packet-level signatures, but they also include protocol data besides the length and direction metadata. We argue that such semantically meaningful data are more adapted for accurate traffic signatures; we leverage similar ones in our framework from Chapter 6 as well. Besides, as the work *IoTA* by Duan *et al.* [26] is intended to be leveraged for intrusion detection, its accuracy should be maximized. Therefore, they implemented an error-recovery mechanism, to allow for traffic which sporadically departs from their model, while staying similar their model.

The goal of our framework described in Chapter 6 is also the profiling of SH events. The difference lies in that, instead of packets, it works with what we have dubbed Flow IDs, a concept analogous to traffic flows in that they are an abstraction of packets, which only consider the relevant attributes, and they are used to represent a set or sequence of packets sharing those attributes. In theory, our framework could be easily adapted to consider individual packets instead of flows. However, converting the nodes in the event signature tree from the current Flow IDs (which aggregate several packets) to single packets would entail two main issues. First, the tree would obviously become tremendously large, which would render the framework's operation intractable. Besides, the current strategy of iterative firewalling works well

with flows, as all packets corresponding to the flow are blocked, which completely obstructs the communication. On the other hand, blocking only a single packet has virtually no effect: if the carried data was important, the packet might just be retransmitted; if not, the loss may simply be ignored. This strategy could therefore not progress past the first level of the tree.

(b) Object: Device; Behavior: Packets

We increase the object abstraction by one stage, and now consider the profiling of a complete device, based on the packets it issues. This is one of the cells that our profile format described in Chapter 5 is intended to cover.

Few works exist which have attempted to automatically generate a signature for a full device based on singular packets. While a single event might already involve a large amount of different packets, a device might issue numerous such events. It becomes practically laborious to profile a complete device at the fine-grained level of packets. We argue that a more practical approach, which does not lose substantial information, is to profile a device at the level of its flows, which aggregate packets sharing select attributes, as described in Section (e).

Given a specific device, the aforementioned *IoTA* starts by producing packet-level models for each of the observed events. To reduce the size taken by the set of all models, they designed an approach to merge events exhibiting an equivalent subset of packets into a single data structure. The latter takes the form of a directed graph, with each node representing a packet. When the events diverge in terms of packets, the graph branches into multiple branches which correspond to possible subsequent packets, each representing a different event. The set of all of these structures provides a more global model of the device's complete operation. Still, as they built each event signature by triggering the event and recording the subsequent traffic, their approach is not able to model non-triggered communication patterns, namely those we dubbed periodic and aperiodic in Section 6.2. Because of this, their device model overlooks a substantial part of the device's complete operation.

(c) Object: Network; Behavior: Packets

The last cell on the row of packet-level profiling concerns the SH network as a whole. The main draw of our format from Chapter 5 is to provide support for interactions involving different devices, which means it is, by extension, intended to model the network as a whole. Therefore, it also covers this cell.

Works related to the previous approach of packet-level profiling of devices were already scarce, due to its tediousness. The profiling of the network as a whole, which lies at an increased level of abstraction, would again be sub-

stantially more tedious; doing it automatically would really be infeasible in practice. To the best of our knowledge, no work has attempted this.

(d) **Object: Event; Behavior: Flows**

We switch to the next row and move from profiling based on individual packets to that based on flows. Recall that a *flow* is basically an abstraction of packets, exhibiting only select attributes, which are used as an aggregate of a (limited) set of packets sharing those attributes. The typical attributes considered are the source and destination hosts and ports as well as the transport protocol (e.g. TCP or UDP), which form what is called the network *5-tuple*.

This cell is the one covered by our framework described in Chapter 6. In the context of that work, we defined a representation of flows, that we called the *Flow ID*, which augments the aforementioned *5-tuple* with additional features, most importantly application-layer protocol-specific ones. The basic objective is profiling the given SH events, by extracting the associated Flow IDs. Similarly to the packet-level profiling works, our strategy is to trigger the event under scrutiny several times, record its associated traffic, and apply the extraction logic. Our main contribution with this framework is that we designed a methodology to do so in an in-depth way, allowing to discover substantially more flows, including many *hidden* ones missed by existing work.

Mandalari *et al.* [314] applied a similar strategy. However, their objective was different, namely identifying the domain names required for a SH event's correct operation, and separate those from the non-required ones. [314]. While they managed to extract several domain names, their strategy is more shallow than ours.

The *BehavIoT* tool by Hu *et al.* [346] is split in two stages. The first consists in modeling the behavior of an SH event based on its network flows. Their tool is able to identify two of the three categories of events specified in Section 6.2, namely the triggered and periodic ones. By process of elimination, all communication which does not belong to any of those two can be assigned to the last one, that of aperiodic events. Once the events have been profiled, they leverage them as building blocks to model the whole system, as described further in Section (i).

We argue this level of abstraction is the good compromise to enable an in-depth profiling like we did in Chapter 6. However, our approach inherently only covers one of the three categories of communication designated in Section 6.2, namely the triggered communication. It overlooks the two other, the periodic and aperiodic ones, which occur without any input from the user. Our framework could be tweaked to account for the periodic one as well. For those, instead of recording traffic only when a specific event is performed, we could do so continuously, then apply periodic models similar

to those in *BehavIoT* [346] to identify Flow IDs pertaining to periodic events. The construction of the event signature tree can then progress by applying the selective Flow ID firewalling on the identified periodic flows, and continuing this workflow recursively. Nevertheless, this approach can never ensure that all periodic events have been identified; regardless of the capture duration, events with a very long period might be missed.

(e) Object: Device; Behavior: Flows

Profiling a complete device with the help of its flows is practically more doable than doing this with individual packets. This level of abstraction is basically what MUD was designed for, and is one also supported by our profile format. In the context of our work, Gallez and Zhang [359], with their master's thesis, developed a tool able to extract simple traffic patterns, *e.g.* a pair of an ARP request and response, or the four DHCP messages exchanged for IP address allocation, from a given PCAP file, and produce a profile, complying with our format, including those.

Barrera *et al.* [21], along their initial proposition for a device format similar to MUD, introduced a workflow to automatically generate a profile complying with their format. The traffic of the device is simply recorded during a period, then all observed traffic involving the device is considered as part of its behavior, and added to the profile.

Hamza *et al.* [364] took basically the same approach to develop their system which automatically produces a MUD profile for a device based on its network traffic. Nevertheless, since MUD supports specifying a host's domain name instead of its IP address, they improved the approach by adding logic to map the observed IP addresses to their associated domain name. As discussed in Section 6.6.6.3, this enables more robust profiles.

(f) Object: Network; Behavior: Flows

This is the final cell which is covered by our profile format. Indeed, it is able to express interactions between multiple devices, by specifying the different flows coming into play.

As for the same based on single packets, profiling of the whole network based on flows proves complicated in practice. While approaches exist to do so for individual events, it remains an open question to know if a cross-device interaction can be modeled based on the traffic from its constitutive events.

(g) Object: Event; Behavior: Event

We go to the last row, which steps away from network traffic for the building blocks of the profiling. Instead, this row leverages the SH events themselves

as profiling units.

The first cell is trivial, as it involves profiling a SH event based on its constitutive events. Obviously, this is a tautology: a SH event completely describes itself.

(h) Object: Device; Behavior: Events

This cell has us asking how to profile a singular SH device, based on its events. More precisely, it consists in extracting the events that a given SH device is able to perform.

This level of behavioral abstraction is inherently very different from the two previous ones. While the latter both observed the network traffic, albeit at a differing granularity, this one completely abstracts network traffic behind the events. Therefore, the approaches to profiling must be radically different. A popular one is to leverage the textual description of the device itself (e.g. in its manual) or of its companion app (e.g. in the related app store, namely Google Play Store [384] for Android, or the iOS App Store [385]). Natural Language Processing (NLP) techniques can be applied on those textual descriptions to extract the supported events.

Sun *et al.*'s *D-interact* [27] was already mentioned in Section (a), as they extract packet-level signatures of individual events. Before this step, though, they apply a strategy to identify said events by leveraging such an NLP technique on the devices' textual functional descriptions.

(i) Object: Network; Behavior: Events

The final, and most abstract approach to SH profiling is that of the SH network as a whole, using the events which take place in it. As discussed in Section 5.6.3, one single event might be translated to numerous traffic patterns in practice. This more abstract approach prevents a potential combinatorial explosion when combining several events, all related to multiple patterns.

The second stage in the operation of Hu *et al.*'s work *BehavIoT* [346] is the modeling of interactions occurring in a given SH system, based on the constitutive events extracted at the previous step (described in Section (d)). The model takes the form of a PFSM where states are the events, and transitions go from one event to another with a given probability. The example given in their paper is a voice command issue towards an Amazon Echo Spot device which entails either a smart plug to switch either on or off, with approximately 50% probability each. As the model takes the form of a PFSM, it can easily be leveraged for subsequent anomaly detection. Their model is highly expressive, but is *ad hoc* to the specific network it was generated for.

HAWatcher, by Fu *et al.* [322] applies an NLP approach, taking as input the description of SH abstract properties tied to given devices from the Smart-

Things platform [299]. They mine the correlations between device properties (e.g. a smart heater heating a room) and physical properties (e.g. the room temperature). Finally, they draw a map of the potential interactions between devices which act on or react to the same physical properties.

Sun *et al.*'s *D-interact* [27] is yet again present in this cell, which demonstrates the wide scope of their contribution. After having identified the events supported by each device thanks to an NLP approach, and having extracted a packet-level signature for said events, their system generates a model of the complete network. In the same vein as the aforementioned *HAWatcher*, they produce a graph of all discovered interactions. While *HAWatcher* only supported those enabled through intermediary physical channels, *D-interact* also considers those configured via a home automation app.

7.1.1 Note about data purity

Taking a step back, it is worth considering whether a *completely automated* creation of profiles based on traffic captures is actually desirable. Research works which focus on deriving a model of legitimate behavior based on data, often assume that the initial data is free from any malicious content. However, attackers could manipulate said initial captures to include malicious patterns, which would then end up in the profiles, and be considered as legitimate for further operation. This concern is similar in concept as the one about *data poisoning attacks*, in the field of Machine Learning.

7.2 Model of the event failure probability

In Section 6.6.3, we modeled P_e , the probability that an event would fail, given F_e , the set of Flow IDs blocked by the firewall, as Equation 6.6. This equation leverages a Noisy-OR model for the contribution of each blocked Flow ID $f_i \in F_e$ to the global probability. In order to use this model, we assume that each Flow ID f_i has an independent effect on the event failure.

In practice, this assumption might prove simplistic, as some Flow IDs might have no effect by themselves, but contribute to an event failure when combined with others, which would disqualify the individual effect assumption. Besides, some Flow IDs only appear when other ones (their parent in the tree) are blocked. This indicates a dependency of those Flow IDs on their parent, which, again, goes against the assumption. To illustrate such a dependency between a parent and a child Flow ID, we take back the example event signature tree presented in Section 6.4.2, generated by applying the framework, with the *path pruning* heuristic, over the Tapo P110 smart plug. The Flow ID labeled as ③ on Figure 6.6a occurs only upon blocking the one labeled as ②. A similar observation can be made about the Flow IDs ④ and ⑤,

which only appear when blocking both Flow IDs ① and ②.

A way to relax this assumption would be to leverage a more general model to express the causal strength of each blocked Flow ID. One of those is an extension of the basic Noisy-OR model described before, named the Recursive Noisy-OR (RNOR) [386]. Figure 7.1 illustrates a comparison between the operation of the basic Noisy-OR model and the extended RNOR. While the basic Noisy-OR combines all variables in a single OR operation, the recursive version models hierarchy by allowing multiple layers of OR gates, each combining a subset of the initial variables. The intermediate results are recursively combined until the model's last layer, where the final probability is given. This hierarchical modeling allows expressing causes, *i.e.* blocked Flow IDs, having a joint effect on the outcome probability, which is dubbed as *synergy*.

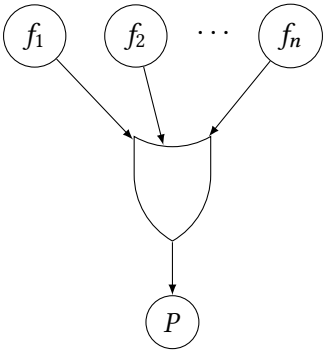
We will illustrate synergy by taking again the *toggle* event of the Tapo P110 smart plug as example. Performing this event requires various communication patterns; one of them can be described abstractly as the communication between the companion app (on the phone) and the plug itself. The default way to do so is for the phone to directly contact the plug using HTTP. As described in Section 6.4.2, if this pattern cannot succeed, the app can alternatively contact the remote server with name `*-app-server.iot.i.tplinkcloud.com`, using HTTPS. If either of those patterns is allowed, the event will succeed; but if they are both blocked, the event will most likely fail. Those two patterns *synergize* together, and could be modeled as such in a RNOR model, whereas their relation is missed in a plain Noisy-OR model. While the RNOR model is more expressive, it also requires domain knowledge to be modeled accordingly, which is not trivial to obtain from our event signature trees, and would require extensive further analysis.

7.3 Geolocation-based traffic divergences

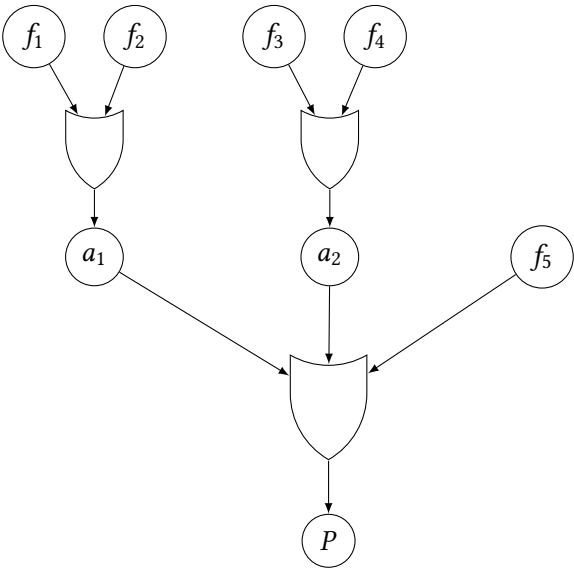
The objective of framework described in Chapter 6 is to exhaustively profile the behavior of SH devices, in terms of their communication patterns. Yet, it overlooks an important parameter that might substantially alter their traffic: their geographical location. Indeed, the actual location of a device might have a significant impact on the traffic characteristics, most notably the domain names contacted. In the following, we discuss the reasons for such traffic divergences, and close the discussion by giving insights on how to consider them during our framework's operation.

7.3.1 Contacting the closest servers

The main geolocation-based difference that may arise in network traffic is the servers which are contacted by the device at hand. Numerous companies



(a) Basic Noisy-OR



(b) Recursive Noisy-OR

Figure 7.1: Comparison between basic Noisy-OR and Recursive Noisy-OR models

provide servers scattered all around the world, such that they are close to their customers, wherever they are on the planet.

Various strategies exist to provide such functionality in the device firmware, which can be used conjointly. The most straightforward is to produce region-based firmware, which will be shipped only in the device targeted towards a specific region's market. Those would be configured upstream with the names of the regional servers. While being the simplest one, this strategy requires more fine-grained configuration of the firmware, which increases the time to market of the devices. Besides, it lacks flexibility, as the servers' names are "hard-coded" in the firmware: if the names must be changed, a firmware update is required on all devices.

An alternative strategy is the use of *load-balancing* across servers based on their geographical location. With this strategy, all devices contact the same domain name, regardless of their location, but the domain name will be resolved to different actual servers. This strategy requires more configuration from the vendor, but also offers more flexibility in the selection of the server receiving the data, in practice often the closest one to the querying device. The upstream firmware configuration is also streamlined, as all regional firmware may now contact the same domain name.

Example For its operation, our TP-Link HS110 smart plug contacts the API server with domain name `euw1-api.tplinkra.com`. The "euw" part of the name suggests this is the server located in **Europe West**. Two other API servers exist, with the following domain names:

- `use1-api.tplinkcloud.com`: United States & America
- `aps1-api.tplinkcloud.com`: Asia-Pacific

While this information is quite hidden by TP-Link, they released the list of domain names contacted by the *Omada* series of devices [387]. The aforementioned domains are not used by the Omada devices, but a similar naming scheme is used, suggesting the scheme is implemented across all devices manufactured by TP-Link.

7.3.2 Regional privacy rules

It is widely known that manufacturers of SH devices collect personal data about their users [18, 19]. To leverage data collection, manufacturers usually configure the devices to communicate with servers besides the ones used solely for the devices' operation, and to transfer them the user's personal data.

This research has been pursued in Europe, a region which is notoriously highly concerned with the privacy of users' personal data. The flagship example is the General Data Protection Regulation (GDPR) [388], a legal measure

introduced by the European Union, which aims at protecting the European citizens' privacy by restricting the usage of their personal data. Therefore, manufacturers should not collect as much data related to European customers compared to other regions.

In practice, this difference in the processing of personal data might be translated in differing communication patterns, depending on the region where a device is deployed. Whilst in other regions, devices could contact the aforementioned optional tracking servers, in Europe, such traffic would be completely suppressed.

7.3.3 Implementation insights

To wrap up the discussion about communication patterns varying with the geolocation of devices, we propose insights on how our framework could be enhanced to allow discovering patterns issued in different regions.

There are two general ways to trigger region-based specific traffic. The first, which is the one requiring less resources, is simply to modify the location setting in a device's companion app. For example, this is possible with the Xiaomi camera [292] present in our testbed. Notably, to unlock some streaming capabilities, we had to set our region to "Chinese mainland" for our experiments, which is yet another example of differing functionality from one region to another. This solution, while simple, is not guaranteed to give the expected results: the app might still use the phone's location to configure its and the device's functionality.

A more thorough solution, albeit being heavier in configuration and resources, is the use of a Virtual Private Network (VPN) to virtually relocate the testbed network to another location. This solution requires renting servers in all the targeted regions, which will act as the VPN endpoints, spoofing the region where the device and companion app are actually located.

In order to allow our framework to automatically extract communication patterns from different regions, the framework's execution on each event should be extended by implementing a switch to every new location, regardless of the chosen solution. Let n be the total number of targeted regions, the execution time of the framework would be multiplied by a factor of n . As the execution already took several days for most events (when using the *path pruning* heuristic), the framework's operation might become intractable if numerous regions are configured.

An alternative would be to deploy n instances of the testbed, each tied to a specific region only. As one testbed can only utilize one end device at a time, one would not need a copy of each device in each testbed. Still, at least one AP and one smartphone running the companion apps would be required in each.

7.4 Clear-box approach to profiling

The approach developed in this thesis is, by design, an *opaque-box* approach. Indeed, it considers the device under test as an opaque box, which means we only have access to the input and output of the device, and not to its internal operation. This approach is the most commonly used among related work. The opposite strategy is the *clear-box* approach. When using this approach, one has access to the internal operation of the system, *i.e.* its source code. One can then leverage this knowledge to perform a static analysis of the source code to find relevant information.

In the context of this thesis, more precisely on the topic of SH devices profiling, a clear-box approach using static analysis sounds promising. Indeed, it has, by design, complete access to the behavior of a device, and could therefore exhaustively extract all the existing communication patterns of the device.

The other variant of clear-box approaches is dynamic analysis, which works by executing the obtained code in a sandbox. In our case, it suffers from the same limitation as opaque-box analysis: it is undecidable whether all communication paths have been exerted.

Following this rationale, we explored the approach of clear-box static analysis. However, it proved less fruitful than expected, or even impossible. This section will delve into the details of our attempts, and the roadblocks we encountered.

7.4.1 Device firmware

The most straightforward strategy to apply clear-box profiling would be to obtain the source code for the device's firmware. However, for most commercial SH devices, the firmware is unreachable, for commercial reasons: vendors want to keep it secret to avoid copies. Extracting the firmware as a binary file might be possible, but it would require access to a specific pin in the device's circuit board, which necessitates dismantling the device. Even then, such a pin might not be accessible. Anyway, this binary firmware is most often encrypted and/or obfuscated. Obtaining a clear version of it by decompiling it is very complex, and might be a research topic in itself.

7.4.2 Companion app APK

A more promising, but less straightforward strategy would be to try obtaining the source code for the device's companion app. A caveat of this approach is that it would only be able to identify the communication patterns issued by or to the app itself, and would therefore miss the ones which only involve the device and not the app. On Android smartphones, applications are

packaged as APKs; those are essentially Java Archive (JAR) files specific to Android. The reverse engineering of an APK goes through two subsequent stages: *disassembly* and *decompilation*. The former consists in obtaining a sort of assembly code from the APK, expressed in the Smali [389] format. The latter tries converting the Smali assembly code to the initial Java code. Naturally, this is not possible to achieve totally, for the same reasons as for the firmware; nevertheless, as the language and structure of an APK is generally more standard and similar to other ones, it is easier to perform reverse engineering on an APK than a firmware. Disassembly and decompilation of APK has been heavily studied in the past, thanks to the popularity of Java and Android. Powerful tools exist, namely *apktool* [390] for disassembly, and *JADX* [391] for decompilation.

Obtaining the source code is the starting point to try extracting information in a clear-box fashion. The next one is to actually find the desired information with static analysis. With *IoT Athena* [367], Wan *et al.* perform a lightweight clear-box analysis to identify all interactions a user can perform with the app, which embody the available SH events supported by the device. To do so, they extract the file `AndroidManifest.xml` from the APK by disassembling it. This file, among others, describes all the activities provided by an app. However, they do not further analyze the APK.

In our case, we want to identify the network behavior of the companion app, which means describing its communication patterns in terms of network-related features, *i.e.* hosts, protocols, ports, etc. Broadly, this requires finding the functions in the source code which perform the network communication, as well as their arguments, as both of those entities provide information about the communication. We will give an example in Java, as most Android APKs are programmed in this language. In Java, the class used to perform operations over a TCP socket is the aptly named `Socket` [392]. Upon instantiation of an object from the `Socket` class, the programmer can provide, as parameters, the hostname and port number to connect to. In this case, both the method itself (`Socket`'s constructor) and the parameters carry meaningful data: the former gives the protocol used (TCP) while the latter give the host and port number. Wang *et al.* [393] apply a limited variant of such an analysis, but mostly resort to symbolic execution, which is a dynamic analysis strategy. Kim *et al.* [394] apply static analysis on an Android app's binary, but only identify request/response patterns related to HTTP(s) communication.

In order to find this information from the source code, we considered the concept of *data flow analysis*: in a nutshell, this strategy consists in following the flow of data (*e.g.* variables) inside a program, from their *source* to their *sink*. In our case, a source provides network-related data to a program, *e.g.* a hostname or port, while a sink uses such data for network communication, *e.g.* a `Socket`. The generic workflow of data flow analysis consists in identifying

the relevant sinks, then performing backtracking to discover the sources.

Due to the size of an APK's source code, the former is intractable manually. Rasthofer *et al.* designed SuSi [395], an ML-based tool which is able to extract data sources and sinks from a given Android API. Piskachev *et al.* proposed SWAN [396], an upgrade to the aforementioned tool which detects more methods and gives enriched information as output. In both cases, the authors applied the tool over Android 4.2 as an example, which extracted numerous source and sink functions found in the Android API, including ones relevant for network communication.

Regarding backtrack analysis, Several researchers have designed strategies and implemented tools to perform it. Androguard [397] is a full-fledged Python tool which disassembles and decompiles an APK, and is notably able to reconstruct the call graph starting from a given function, *i.e.* find the sequence of function calls up to the given one. However, it only extracts the type of the parameters, and not their value, which greatly limits the extracted information. For instance, an argument carrying a port would only be labeled as an integer, without giving the actual port number. Hoffmann *et al.* proposed Static Android Analysis Framework (SAAF) [398], which applies backtrack analysis over Smali code directly, skipping the need for decompilation. While it gives additional information compared to Androguard, it still falls short in extracting the parameters' values. Finally, Arzt *et al.* proposed FLOWDROID [399], a tool tailored to identify data leaks in a given APK, which is not relevant for our use case.

7.4.3 Takeaways

We pursued exploratory research in trying to extract the network behavior of SH devices and their companion apps by applying a clear-box approach, leveraging their source code. While this seemed theoretically possible for the app, the current state of the related research is not developed enough to identify precisely the app's network-related characteristics. We deem this a promising future research direction, as the use case is motivating and important in the current landscape. Regarding the device firmware itself, obtaining a usable version of its code is already a big challenge in itself, which considerably hinders this approach.

Since several years, our society is undergoing an unavoidable transition towards an ever increasingly digitized world. This transformation started with the emergence of the Internet in our daily lives, and is now comparable to that of the industrial revolution, so much so that specialists have coined the term “Industry 4.0” to describe the industries which make use of IoT technologies. Nonetheless, industries are far from the only sector affected by such technologies; agriculture, urban areas, the health sector were also metamorphosed, as well as the focus of this thesis, households, which became Smart Homes.

The ubiquity of such small computing devices poses critical concerns related to security and privacy. As a user, you probably would not want your system to be hacked, or your private data to leak for business purposes. To illustrate with very concrete examples, the former might make it possible for a hacker to unlock your front door’s smart lock, while the latter makes your phone number available for advertisers to call you unexpectedly. In Europe, it has even become a legal concern, thanks to the NIS2 [12] and GDPR [388] directives, respectively regarding security and privacy.

With this thesis, we take a step towards more secure Smart Home networks. It is now time to wrap up this manuscript. We will summarize our main contributions, and the main perspective for improvement for each of those.

We started with a literature-wide survey of public datasets oriented towards research related to IoT network security in general [1]. As we strived to render our compilation easy to browse, it has the potential to alleviate the tedious work of finding a relevant dataset. We hope that it will aid the research community in this way. During our survey, we identified gaps in the available datasets, mainly regarding protocols which are lacking in the datasets. Consequently, we encourage further research projects intending to release datasets involving those protocols, namely LoRa and CoAP among others, and containing sound data.

Afterwards, we designed a language able to express the communication patterns a device is expected to perform [2]. While existing formats only supported singular patterns, ours addresses Research Question 1 by also supporting interactions between multiple devices, which are frequent in practice in order to perform home automation tasks. Such a format can be used to

produce a profile for a device, describing, hopefully exhaustively, the communicating behavior of said device. A security system, *e.g.* a firewall, can then leverage the profile as an anomaly-based security solution. To streamline the usage of our format, we implemented a pipeline which, based on a profile formatted accordingly, produces the necessary configuration files corresponding to the profile, for the widespread Linux firewall *nftables*. In its current state, the profiles and firewall only support communication leveraging protocols from the TCP/IP stack. An interesting extension, though laborious, would be to extend support to other IoT protocols, *e.g.* LoRaWAN, Zigbee, or the novel Matter over Thread which is currently gaining traction.

We followed by addressing the question of the creation of such traffic profiles, that is Research Question 2. While substantial works had tackled the question, they did so in a quite superficial way, only considering the devices' default traffic. On the contrary, we argue that devices might exhibit hidden traffic patterns, potentially missed by existing approaches. For device's profile to be considered complete, such patterns cannot be neglected. Following this rationale, we developed a mostly automated framework able to extract the patterns issued by a device's action, in a more in-depth way than existing work [3]. It does so by performing the action, then recursively blocking the observed patterns to make new ones occur, until the action fails or no new pattern is discovered. This workflow allows uncovering *hidden* traffic patterns which should not occur upon ideal operation, but are still an integral part of the device's expected behavior. Those are useful as a backup communication pattern, when the default one cannot succeed. As such, they embody the device's *robustness*. We assessed Research Question 3 by evaluating all extracted patterns based on empirical metrics, with extra focus on the *hidden* ones. Varying improvement perspectives can be envisioned for this work: extending the scope to cross-device interactions, leveraging other traffic disturbances than a simple binary verdict, or considering traffic divergences due to geolocation.

For an IoT device which exhibits limited and predictable communication patterns, a profile comprising all such patterns has the potential to be a great source of information to secure the device itself and its surrounding hosts. As virtually any network will be home to IoT technology in the near future, the implications regarding its security are huge. However, the exhaustiveness of such a profile is essential to prevent unwanted failures. This thesis took a step in this direction, by providing a profile format able to express more accurately complex traffic patterns of today's SH networks, and a framework to exhaustively extract patterns tied to a SH device's operation. Yet, numerous doors are still open on this path, whether regarding overlooked sources of traffic divergences, or even a completely different approach to profiling leveraging the device's firmware.

Bibliography

- [1] F. De Keersmaecker, Y. Cao, G. K. Ndonda, and R. Sadre. “A Survey of Public IoT Datasets for Network Security Research”. In: *IEEE Communications Surveys & Tutorials* 25.3 (2023), pp. 1808–1840. DOI: 10.1109/COMST.2023.3288942.
- [2] F. De Keersmaecker, R. Sadre, and C. Pelsser. “Supervising Smart Home Device Interactions: A Profile-Based Firewall Approach”. In: *2024 IFIP Networking Conference (IFIP Networking)*. 2024, pp. 413–422. DOI: 10.23919/IFIPNetworking62109.2024.10619760.
- [3] F. De Keersmaecker, R. Van Boxem, C. Pelsser, and R. Sadre. “The Forest Behind the Tree: Revealing Hidden Smart Home Communication Patterns”. In: *2025 IEEE 33rd International Conference on Network Protocols (ICNP)*. 2025, pp. 1–12. DOI: 10.1109/ICNP65844.2025.11192388.
- [4] F. De Keersmaecker, R. Sadre, and C. Pelsser. *Supervising Smart Home Device Interactions: A Profile-Based Firewall Approach*. Poster presented at the 8th Network Traffic Measurements and Analysis Conference (TMA 2024). May 2024.
- [5] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. “Understanding the Mirai Botnet”. In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. ISBN: 978-1-931971-40-9.
- [6] B. Krebs. *Krebs on Security*. Krebs on Security. URL: <https://krebsonsecurity.com> (visited on 10/30/2025).
- [7] *About us | OVHcloud Worldwide*. OVHcloud. URL: <https://www.ovhcloud.com/en/about-us> (visited on 10/30/2025).
- [8] *Domain Name System (DNS)*. Oracle. URL: <https://www.oracle.com/cloud/networking/dns> (visited on 10/30/2025).
- [9] O. Yoachimik and J. Pacheco. *Record-breaking 5.6 Tbps DDoS attack and global DDoS trends for 2024 Q4*. Cloudflare, Jan. 2025. (Visited on 03/23/2026).

- [10] B. Krebs. *DDoS Botnet Aisuru Blankets US ISPs in Record DDoS – Krebs on Security*. Krebs on Security, Oct. 2025. (Visited on 10/13/2025).
- [11] O. Yoachimik and J. Pacheco. *Cloudflare’s 2025 Q3 DDoS threat report – including Aisuru, the apex of botnets*. Cloudflare, Dec. 2025. (Visited on 03/23/2026).
- [12] *Directive (EU) 2022/2555 of the European Parliament and of the Council of 14 December 2022 on measures for a high common level of cyber-security across the Union, amending Regulation (EU) No 910/2014 and Directive (EU) 2018/1972, and repealing Directive (EU) 2016/1148 (NIS 2 Directive)*. Accessed: January 29, 2026. European Union. 2022. URL: <https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:02022L2555-20221227>.
- [13] *The Cyber Resilience Act - Summary of the legislative text*. Accessed: February 12, 2026. European Union. 2025. URL: <https://digital-strategy.ec.europa.eu/en/policies/cra-summary>.
- [14] *40+ Smart Home Technology Facts and Statistics in 2022-2024*. Accessed: January 7, 2026. SwiftBeacon. Jan. 2025. URL: <https://swiftbeacon.com/blogs/news/smart-home-technology-statistics>.
- [15] S. Gibbs. *Hackers can hijack Wi-Fi Hello Barbie to spy on your children*. Accessed: February 12, 2026. The Guardian. Nov. 2015. URL: <https://www.theguardian.com/technology/2015/nov/26/hackers-can-hijack-wi-fi-hello-barbie-to-spy-on-your-children>.
- [16] S. Liao, J. Yan, and L. Cheng. “WIP: Hidden Hub Eavesdropping Attack in Matter-enabled Smart Home Systems”. In: *Workshop on Security and Privacy in Standardized IoT (SDIoTSec)*. 2024.
- [17] A. Girish, T. Hu, V. Prakash, D. J. Dubois, S. Matic, D. Y. Huang, S. Egelman, J. Reardon, J. Tapiador, D. Choffnes, and N. Vallina-Rodriguez. “In the Room Where It Happens: Characterizing Local Communication and Threats in Smart Homes”. In: *Proceedings of the 2023 ACM on Internet Measurement Conference*. IMC ’23. Montreal QC, Canada: Association for Computing Machinery, 2023, pp. 437–456. ISBN: 9798400703829. DOI: 10.1145/3618257.3624830.
- [18] C. Matyszczyk. *Samsung’s warning: Our Smart TVs record your living room chatter*. Accessed: February 12, 2026. CNET. Feb. 2015. URL: <https://www.cnet.com/news/privacy/samsungs-warning-our-smart-tvs-record-your-living-room-chatter>.

- [19] G. A. Fowler. *Alexa has been eavesdropping on you this whole time*. Accessed: February 12, 2026. The Washington Post. May 2019. URL: <https://www.washingtonpost.com/technology/2019/05/06/alexa-has-been-eavesdropping-you-this-whole-time>.
- [20] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics". In: *IEEE Transactions on Mobile Computing* 18.8 (Aug. 2019), pp. 1745–1759. ISSN: 1558-0660. DOI: 10.1109/TMC.2018.2866249.
- [21] D. Barrera, I. Molloy, and H. Huang. "Standardizing IoT network security policy enforcement". In: *Workshop on decentralized IoT security and standards (DISS)*. 2018, p. 6.
- [22] A. Hamza, D. Ranathunga, H. H. Gharakheili, T. A. Benson, M. Roughan, and V. Sivaraman. "Verifying and Monitoring IoTs Network Behavior Using MUD Profiles". In: *IEEE Transactions on Dependable and Secure Computing* 19.1 (2022), pp. 1–18. DOI: 10.1109/TDSC.2020.2997898.
- [23] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky. "Packet-Level Signatures for Smart Home Devices". In: *Proceedings 2020 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2020. ISBN: 978-1-891562-61-7. DOI: 10.14722/ndss.2020.24097. (Visited on 11/07/2023).
- [24] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi. "D²IoT: A Federated Self-learning Anomaly Detection System for IoT". In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 2019, pp. 756–767. DOI: 10.1109/ICDCS.2019.00080.
- [25] Y. Wan, K. Xu, F. Wang, and G. Xue. "Characterizing and Mining Traffic Patterns of IoT Devices in Edge Networks". In: *IEEE Transactions on Network Science and Engineering* 8.1 (2021), pp. 89–101. DOI: 10.1109/TNSE.2020.3026961.
- [26] C. Duan, S. Li, H. Lin, W. Chen, G. Song, C. Li, J. Yang, and Z. Wang. "IoTa: Fine-Grained Traffic Monitoring for IoT Devices via Fully Packet-Level Models". In: *IEEE Transactions on Dependable and Secure Computing* 01 (Dec. 2023). Publisher: IEEE Computer Society, pp. 1–17. ISSN: 1545-5971. DOI: 10.1109/TDSC.2023.3340563. (Visited on 12/13/2023).

- [27] M. Sun, K. Li, Y. Zheng, W. Zhang, H. Li, and L. Sun. "Inferring Device Interactions for Attack Path Discovery in Smart Home IoT". In: *Wireless Algorithms, Systems, and Applications*. Ed. by L. Wang, M. Segal, J. Chen, and T. Qiu. Cham: Springer Nature Switzerland, 2022, pp. 64–75. ISBN: 978-3-031-19208-1. DOI: 10.1007/978-3-031-19208-1_6.
- [28] K. Rose, S. Eldridge, L. Chapin, et al. "The Internet of Things: An Overview". In: *The internet society (ISOC)* 80.15 (2015), pp. 1–53.
- [29] J. Tournier, F. Lesueur, F. Le Mouël, L. Guyon, and H. Ben-Hassine. "A survey of IoT protocols and their security issues through the lens of a generic IoT stack". In: *Internet of things* 16 (2021), p. 100264.
- [30] H. Mamat, B. Ibrahim, and M. Sulong. "Network Topology Comparison for Internet Communication and IoT Connectivity". In: *2019 IEEE Conference on Open Systems (ICOS)*. 2019, pp. 1–5. DOI: 10.1109/ICOS47562.2019.8975702.
- [31] S. C. Ergen. "ZigBee/IEEE 802.15. 4 Summary". In: *UC Berkeley, September* 10.17 (2004), p. 11.
- [32] D. Rodenas-Herraz, A.-J. Garcia-Sanchez, F. Garcia-Sanchez, and J. Garcia-Haro. "Current Trends in Wireless Mesh Sensor Networks: A Review of Competing Approaches". In: *Sensors* 13.5 (2013), pp. 5958–5995.
- [33] E. Anthi, S. Ahmad, O. Rana, G. Theodorakopoulos, and P. Burnap. "EclipseIoT: A secure and adaptive hub for the Internet of Things". In: *Computers & Security* 78 (2018), pp. 477–490. ISSN: 0167-4048. DOI: <https://doi.org/10.1016/j.cose.2018.07.016>.
- [34] I. Fernando. *What Is a Zigbee Hub? Your Complete Smart Home Guide*. Accessed: March 31, 2026. ZigbeeHubs. Dec. 2025. URL: <https://zigbee-hubs.com/zigbee-hub>.
- [35] D. Diener. *What Is Home Assistant? And Why Isn't Everyone Using It?* Accessed: March 31, 2026. How-To Geek. June 2024. URL: <https://www.howtogeek.com/what-is-home-assistant-and-why-isnt-everyone-using-it>.
- [36] H. Zimmermann. "OSI reference model-the ISO model of architecture for open systems interconnection". In: *IEEE Transactions on communications* 28.4 (1980), pp. 425–432.
- [37] R. T. Braden. *Requirements for Internet Hosts - Communication Layers*. RFC 1122. Oct. 1989. 116 pp. DOI: 10.17487/RFC1122.
- [38] W. Eddy. *Transmission Control Protocol (TCP)*. RFC 9293. Aug. 2022. 98 pp. DOI: 10.17487/RFC9293.

- [39] J. Postel. *Internet Protocol*. RFC 791. Sept. 1981. 51 pp. DOI: 10.17487/RFC0791.
- [40] J. Iyengar and M. Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. RFC Editor, May 2021. DOI: 10.17487/RFC9000.
- [41] O. Bonaventure. *Computer Networking : Principles, Protocols and Practice, third edition*. 2019.
- [42] D. Waitzman. *Standard for the transmission of IP datagrams on avian carriers*. RFC 1149. Apr. 1990. 2 pp. DOI: 10.17487/RFC1149.
- [43] S. E. Deering and B. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 1883. RFC Editor, Dec. 1995. 37 pp. DOI: 10.17487/RFC1883.
- [44] T. Hu, D. J. Dubois, and D. Choffnes. “IoT Bricks Over v6: Understanding IPv6 Usage in Smart Homes”. In: *Proceedings of the 2024 ACM on Internet Measurement Conference*. IMC ’24. Madrid, Spain: Association for Computing Machinery, 2024, pp. 595–611. ISBN: 9798400705922. DOI: 10.1145/3646547.3688457.
- [45] J. Postel. *Internet Control Message Protocol*. RFC 792. Sept. 1981. 21 pp. DOI: 10.17487/RFC0792.
- [46] B. Fenner. *Internet Group Management Protocol, Version 2*. RFC 2236. Nov. 1997. 24 pp. DOI: 10.17487/RFC2236.
- [47] B. Haberman. *Internet Group Management Protocol, Version 3*. RFC 9776. Mar. 2025. 48 pp. DOI: 10.17487/RFC9776.
- [48] D. C. Plummer. *An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*. RFC 826. Nov. 1982. 10 pp. DOI: 10.17487/RFC0826.
- [49] J. Postel. *User Datagram Protocol*. RFC 768. Aug. 1980. 3 pp. DOI: 10.17487/RFC0768.
- [50] R. Droms. *Dynamic Host Configuration Protocol*. RFC 2131. Mar. 1997. 45 pp. DOI: 10.17487/RFC2131.
- [51] J. Martin, J. Burbank, W. Kasch, and D. L. Mills. *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905. June 2010. 110 pp. DOI: 10.17487/RFC5905.
- [52] *Amazon EC2 instance IP addressing*. Accessed: March 31, 2026. Amazon Web Services. 2026. URL: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-instance-addressing.html>.

- [53] *Public IP addresses in Azure - Azure Virtual Network* [Microsoft Learn. Accessed: March 31, 2026. Microsoft. 2025. URL: <https://learn.microsoft.com/en-us/azure/virtual-network/ip-services/public-ip-addresses>.
- [54] P. Mockapetris. *Domain names - implementation and specification*. RFC 1035. Nov. 1987. 55 pp. DOI: 10.17487/RFC1035.
- [55] S. Cheshire and M. Krochmal. *Multicast DNS*. RFC 6762. Feb. 2013. 70 pp. DOI: 10.17487/RFC6762.
- [56] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications". In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2347–2376. DOI: 10.1109/COMST.2015.2444095.
- [57] R. T. Fielding, M. Nottingham, and J. Reschke. *HTTP Semantics*. RFC 9110. June 2022. 194 pp. DOI: 10.17487/RFC9110.
- [58] T. Berners-Lee, R. T. Fielding, and L. M. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. Jan. 2005. 61 pp. DOI: 10.17487/RFC3986.
- [59] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. 160 pp. DOI: 10.17487/RFC8446.
- [60] E. Rescorla. *HTTP Over TLS*. RFC 2818. May 2000. 7 pp. DOI: 10.17487/RFC2818.
- [61] S. Albright, P. J. Leach, Y. Gu, Y. Y. Goland, and T. Cai. *Simple Service Discovery Protocol/1.0*. Internet-Draft draft-cai-ssdp-v1-03. Internet Engineering Task Force, Nov. 1999. 18 pp.
- [62] F. De Keersmaecker and N. van de Walle. "Using LoRaWAN and Wi-Fi for smart city monitoring in Louvain-la-Neuve". Prom.: Yves Deville, Ramin Sadre. MA thesis. UCLouvain, 2021.
- [63] *Frequency ranges for global or regional harmonization of short-range devices*. Accessed: March 31, 2026. International Telecommunication Union. 2018. URL: https://www.itu.int/dms_pubrec/itu-r/rec/sm/R-REC-SM.1896-1-201809-I!!PDF-E.pdf.
- [64] K. Sairam, N. Gunasekaran, and S. R. Redd. "Bluetooth in wireless communication". In: *IEEE Communications Magazine* 40.6 (2002), pp. 90–96.
- [65] A. F. Harris III, V. Khanna, G. Tuncay, R. Want, and R. Kravets. "Bluetooth low energy in dense IoT environments". In: *IEEE Communications Magazine* 54.12 (2016), pp. 30–36.

- [66] J. A. Gutiérrez. “On the use of IEEE Std. 802.15. 4 to enable wireless sensor networks in building automation”. In: *International Journal of Wireless Information Networks* 14.4 (2007), pp. 295–301.
- [67] U. Raza, P. Kulkarni, and M. Sooriyabandara. “Low Power Wide Area Networks: An Overview”. In: *IEEE Communications Surveys & Tutorials* 19.2 (2017), pp. 855–873.
- [68] C. Milarokostas, D. Tsolkas, N. Passas, and L. Merakos. “A Comprehensive Study on LPWANs With a Focus on the Potential of LoRa/LoRaWAN Systems”. In: *IEEE Communications Surveys & Tutorials* 25.1 (2023), pp. 825–867. DOI: 10.1109/COMST.2022.3229846.
- [69] J. Petajajarvi, K. Mikhaylov, A. Roivainen, T. Hanninen, and M. Pet-tissalo. “On the coverage of LPWANs: range evaluation and channel attenuation model for LoRa technology”. In: *2015 14th international conference on its telecommunications (itst)*. IEEE. 2015, pp. 55–59.
- [70] J. Haxhibeqiri, E. De Poorter, I. Moerman, and J. Hoebeke. “A survey of LoRaWAN for IoT: From technology to application”. In: *Sensors* 18.11 (2018), p. 3995.
- [71] A. Lavric, A. I. Petrariu, and V. Popa. “SigFox Communication Protocol: The New Era of IoT?”. In: *2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI)*. IEEE. 2019, pp. 1–4.
- [72] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer. “Overview of cellular LPWAN technologies for IoT deployment: Sigfox, LoRaWAN, and NB-IoT”. In: *2018 IEEE international conference on pervasive computing and communications workshops (percom workshops)*. IEEE. 2018, pp. 197–202.
- [73] M. B. Yassein, W. Mardini, and A. Khalil. “Smart homes automation using Z-wave protocol”. In: *2016 International Conference on Engineering & MIS (ICEMIS)*. IEEE. 2016, pp. 1–6.
- [74] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550. RFC Editor, Mar. 2012. 157 pp. DOI: 10.17487/RFC6550.
- [75] G. Mulligan. “The 6LoWPAN architecture”. In: *Proceedings of the 4th workshop on Embedded networked sensors*. 2007, pp. 78–82.
- [76] “ZigBee IP: an open standard for IPv6-based wireless mesh networks”. In: (Mar. 2013). Accessed: May 7, 2025.
- [77] I. Unwala, Z. Taqvi, and J. Lu. “Thread: An IoT Protocol”. In: *2018 IEEE Green Technologies Conference (GreenTech)*. 2018, pp. 161–167. DOI: 10.1109/GreenTech.2018.00037.

- [78] *Matter Specification Version 1.4*. Connectivity Standards Alliance, Nov. 2024.
- [79] D. Soni and A. Makwana. "A survey on MQTT: a protocol of Internet of Things (IoT)". In: *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*. Vol. 20. 2017, pp. 173–177.
- [80] B. Mishra and A. Kertesz. "The Use of MQTT in M2M and IoT Systems: A Survey". In: *IEEE Access* 8 (2020), pp. 201071–201086. doi: 10.1109/ACCESS.2020.3035849.
- [81] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. "The many faces of publish/subscribe". In: *ACM computing surveys (CSUR)* 35.2 (2003), pp. 114–131.
- [82] Z. Shelby, K. Hartke, and C. Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. RFC Editor, June 2014. 112 pp. DOI: 10.17487/RFC7252.
- [83] "What is Modbus and How does it work?" Accessed: February 13, 2023. Schneider Electric. URL: <https://www.se.com/us/en/faqs/FA168406>.
- [84] "Modbus Messaging on TCP/IP Implementation Guide V1.0b". Accessed: February 13, 2023. Modbus Organization. URL: https://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf.
- [85] M. Tabaa, B. Chouri, S. Saadaoui, and K. Alami. "Industrial Communication based on Modbus and Node-RED". In: *Procedia Computer Science* 130 (2018). The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018) / The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018) / Affiliated Workshops, pp. 583–588. ISSN: 1877-0509. doi: <https://doi.org/10.1016/j.procs.2018.04.107>.
- [86] G. Corotinschi and V. G. Găitan. "Enabling IoT connectivity for Modbus networks by using IoT edge gateways". In: *2018 International Conference on Development and Application Systems (DAS)*. IEEE. 2018, pp. 175–179.
- [87] S.-I. Toc and A. Korodi. "Modbus-OPC UA Wrapper Using Node-RED and IoT-2040 with Application in the Water Industry". In: *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*. IEEE. 2018, pp. 000099–000104.
- [88] "Modbus Application Protocol Specification V1.1b3". Accessed: February 13, 2023. Modbus Organization. URL: https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.

- [89] M. Lauridsen, I. Z. Kovacs, P. Mogensen, M. Sorensen, and S. Holst. "Coverage and Capacity Analysis of LTE-M and NB-IoT in a Rural Area". In: *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*. 2016, pp. 1–5. DOI: 10.1109/VTCFall.2016.7880946.
- [90] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi, and M. Mustaqim. "Internet of things (IoT) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5G-IoT scenarios". In: *IEEE Access* 8 (2020), pp. 23022–23040.
- [91] P. Asghari, A. M. Rahmani, and H. H. S. Javadi. "Internet of Things applications: A systematic review". In: *Computer Networks* 148 (2019), pp. 241–261. ISSN: 1389-1286. DOI: <https://doi.org/10.1016/j.comnet.2018.12.008>.
- [92] J. Yick, B. Mukherjee, and D. Ghosal. "Wireless sensor network survey". In: *Computer Networks* 52.12 (2008), pp. 2292–2330. ISSN: 1389-1286. DOI: <https://doi.org/10.1016/j.comnet.2008.04.002>.
- [93] C. Yin, Z. Xiong, H. Chen, J. Wang, D. Cooper, and B. David. "A literature survey on smart cities". In: *Science China Information Sciences* 58.10 (2015), pp. 1–18.
- [94] C. Li, J. Wang, S. Wang, and Y. Zhang. "A review of IoT applications in healthcare". In: *Neurocomputing* 565 (2024), p. 127017. ISSN: 0925-2312. DOI: <https://doi.org/10.1016/j.neucom.2023.127017>.
- [95] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund. "Industrial Internet of Things: Challenges, Opportunities, and Directions". In: *IEEE transactions on industrial informatics* 14.11 (2018), pp. 4724–4734.
- [96] W. Trappe, R. Howard, and R. S. Moore. "Low-Energy Security: Limits and Opportunities in the Internet of Things". In: *IEEE Security & Privacy* 13.1 (2015), pp. 14–21. DOI: 10.1109/MSP.2015.7.
- [97] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao. "A Survey on Security and Privacy Issues in Internet-of-Things". In: *IEEE Internet of Things Journal* 4.5 (2017), pp. 1250–1258. DOI: 10.1109/JIOT.2017.2694844.
- [98] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella. "IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices". In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 8182–8201. DOI: 10.1109/JIOT.2019.2935189.

- [99] M. T. Paracha, D. J. Dubois, N. Vallina-Rodriguez, and D. Choffnes. "IoTLS: understanding TLS usage in consumer IoT devices". In: *Proceedings of the 21st ACM Internet Measurement Conference*. IMC '21. Virtual Event: Association for Computing Machinery, 2021, pp. 165–178. ISBN: 9781450391290. DOI: 10.1145/3487552.3487830.
- [100] D. Y. Huang, N. Apthorpe, F. Li, G. Acar, and N. Feamster. "IoT Inspector: Crowdsourcing Labeled Network Traffic from Smart Home Devices at Scale". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4.2 (June 2020), 46:1–46:21. DOI: 10.1145/3397333. (Visited on 01/02/2024).
- [101] G. Baran. *Massive IoT Data Breach Exposes 2.7 Billion Records, Including Wi-Fi Passwords*. Accessed: April 1, 2026. Cyber Security News. Feb. 2025. URL: <https://cybersecuritynews.com/massive-iot-data-breach>.
- [102] R. Bodenheim, J. Butts, S. Dunlap, and B. Mullins. "Evaluation of the ability of the Shodan search engine to identify Internet-facing industrial control devices". In: *International Journal of Critical Infrastructure Protection* 7.2 (2014), pp. 114–123.
- [103] H. Al-Alami, A. Hadi, and H. Al-Bahadili. "Vulnerability scanning of IoT devices in Jordan using Shodan". In: *2017 2nd International Conference on the Applications of Information Technology in Developing Renewable Energy Processes & Systems (IT-DREPS)*. 2017, pp. 1–6. DOI: 10.1109/IT-DREPS.2017.8277814.
- [104] B. Tushir, H. Sehgal, R. Nair, B. Dezfouli, and Y. Liu. *The Impact of DoS Attacks on Resource-constrained IoT Devices: A Study on the Mirai Attack*. 2021. arXiv: 2104.09041 [cs.CR].
- [105] M. Selvaraj and G. Uddin. "A Large-Scale Study of IoT Security Weaknesses and Vulnerabilities in the Wild". In: *ACM Trans. Softw. Eng. Methodol.* 34.2 (Jan. 2025). ISSN: 1049-331X. DOI: 10.1145/3691628.
- [106] *Binary classification*. Accessed: April 1, 2026. Wikipedia. Oct. 2025. URL: https://en.wikipedia.org/wiki/Binary_classification.
- [107] V. Bukac, P. Tucek, and M. Deutsch. "Advances and Challenges in Standalone Host-Based Intrusion Detection Systems". In: *International Conference on Trust, Privacy and Security in Digital Business*. Springer. 2012, pp. 105–117.
- [108] G. Fernandes Jr, J. J. Rodrigues, L. F. Carvalho, J. F. Al-Muhtadi, and M. L. Proença Jr. "A comprehensive survey on network anomaly detection". In: *Telecommunication Systems* 70.3 (2019), pp. 447–489.

- [109] B. Mukherjee, L. Heberlein, and K. Levitt. "Network intrusion detection". In: *IEEE Network* 8.3 (1994), pp. 26–41. DOI: 10.1109/65.283931.
- [110] G. Vigna and R. Kemmerer. "NetSTAT: a network-based intrusion detection approach". In: *Proceedings 14th Annual Computer Security Applications Conference (Cat. No.98EX217)*. 1998, pp. 25–34. DOI: 10.1109/CSAC.1998.738566.
- [111] J. Li, Q. Li, S. Zhou, Y. Yao, and J. Ou. "A review on signature-based detection for network threats". In: *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*. 2017, pp. 1117–1121. DOI: 10.1109/ICCSN.2017.8230284.
- [112] P. Szyrkiewicz. "Signature-Based Detection of Botnet DDoS Attacks". In: *Cybersecurity of Digital Service Chains: Challenges, Methodologies, and Tools*. Ed. by J. Kollodziej, M. Repetto, and A. Duzha. Cham: Springer International Publishing, 2022, pp. 120–135. ISBN: 978-3-031-04036-8. DOI: 10.1007/978-3-031-04036-8_6.
- [113] *False positives and false negatives*. Accessed: April 1, 2026. Wikipedia. Feb. 2026. URL: https://en.wikipedia.org/wiki/False_positives_and_false_negatives.
- [114] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. "Anomaly-based network intrusion detection: Techniques, systems and challenges". In: *Computers & Security* 28.1-2 (2009), pp. 18–28.
- [115] M. Eskandari, Z. H. Janjua, M. Vecchio, and F. Antonelli. "Passban IDS: An Intelligent Anomaly-Based Intrusion Detection System for IoT Edge Devices". In: *IEEE Internet of Things Journal* 7.8 (Aug. 2020). Conference Name: IEEE Internet of Things Journal, pp. 6882–6897. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2970501.
- [116] *Stateless vs Stateful Packet Filtering Firewalls*. Accessed: April 1, 2026. GeeksforGeeks. Oct. 2025. URL: <https://www.geeksforgeeks.org/computer-networks/stateless-vs-stateful-packet-filtering-firewalls>.
- [117] *Connection Tracking System*. Accessed: April 1, 2026. Netfilter. Apr. 2021. URL: https://wiki.nftables.org/wiki-nftables/index.php/Connection_Tracking_System.
- [118] W. Noonan and I. Dubrawsky. *Firewall fundamentals*. Pearson Education, 2006.
- [119] G. N. Purdy. *Linux iptables Pocket Reference: Firewalls, NAT & Accounting*. O'Reilly Media, Inc., 2004.

- [120] *Welcome to the OpenWrt Project*. Accessed: November 24, 2025. OpenWrt. 2025. URL: <https://openwrt.org>.
- [121] *The netfilter.org "nftables" project*. Accessed: July 16, 2025. Netfilter. July 2025. URL: <https://www.netfilter.org/projects/nftables/index.html>.
- [122] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung. "Intrusion detection system: A comprehensive review". In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 16–24. ISSN: 1084-8045. DOI: <https://doi.org/10.1016/j.jnca.2012.09.004>.
- [123] M. M. Rahman, S. Al Shakil, and M. R. Mustakim. "A survey on intrusion detection system in IoT networks". In: *Cyber Security and Applications* 3 (2025), p. 100082.
- [124] A. S. Ashoor and S. Gore. "Difference between Intrusion Detection System (IDS) and Intrusion Prevention System (IPS)". In: *Advances in Network Security and Applications*. Ed. by D. C. Wyld, M. Wozniak, N. Chaki, N. Meghanathan, and D. Nagamalai. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 497–501. ISBN: 978-3-642-22540-6.
- [125] Z. Wang and X. Li. "Intrusion Prevention System Design". In: *Proceedings of the International Conference on Information Engineering and Applications (IEA) 2012*. Ed. by Z. Zhong. London: Springer London, 2013, pp. 375–382. ISBN: 978-1-4471-4847-0.
- [126] M. Roesch et al. "Snort: Lightweight intrusion detection for networks". In: *Lisa*. Vol. 99. 1. 1999, pp. 229–238.
- [127] *Suricata*. Accessed: February 18, 2026. Open Information Security Foundation (OISF). URL: <https://suricata.io>.
- [128] E. Albin and N. C. Rowe. "A Realistic Experimental Comparison of the Suricata and Snort Intrusion-Detection Systems". In: *2012 26th International Conference on Advanced Information Networking and Applications Workshops*. 2012, pp. 122–127. DOI: 10.1109/WAINA.2012.29.
- [129] *The Zeek Network Security Monitor*. Accessed: February 12, 2026. URL: <https://zeek.org>.
- [130] B. L. Risteska Stojkoska and K. V. Trivodaliev. "A review of Internet of Things for smart home: Challenges and solutions". In: *Journal of Cleaner Production* 140 (2017), pp. 1454–1464. ISSN: 0959-6526. DOI: <https://doi.org/10.1016/j.jclepro.2016.10.006>.
- [131] S. Solaimani, W. Keijzer-Broers, and H. Bouwman. "What we do—and don't-know about the Smart Home: an analysis of the Smart Home literature". In: *Indoor and Built Environment* 24.3 (2015), pp. 370–383.

- [132] A. Chakraborty, M. Islam, F. Shahriyar, S. Islam, H. U. Zaman, and M. Hasan. "Smart Home System: A Comprehensive Review". In: *Journal of Electrical and Computer Engineering* 2023.1 (2023), p. 7616683. DOI: <https://doi.org/10.1155/2023/7616683>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1155/2023/7616683>.
- [133] A. B. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon. "Home automation in the wild: challenges and opportunities". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: Association for Computing Machinery, 2011, pp. 2115–2124. ISBN: 9781450302289. DOI: 10.1145/1978942.1979249.
- [134] M. Asadullah and A. Raza. "An overview of home automation systems". In: *2016 2nd International Conference on Robotics and Artificial Intelligence (ICRAI)*. 2016, pp. 27–31. DOI: 10.1109/ICRAI.2016.7791223.
- [135] S. Balakrishnan, H. Vasudavan, and R. K. Murugesan. "Smart Home Technologies: A Preliminary Review". In: *Proceedings of the 6th International Conference on Information Technology: IoT and Smart City*. ICIT '18. Hong Kong, Hong Kong: Association for Computing Machinery, 2018, pp. 120–127. ISBN: 9781450366298. DOI: 10.1145/3301551.3301575.
- [136] *Routines in SmartThings*. Accessed: April 2, 2026. SmartThings. 2024. URL: <https://support.smarththings.com/hc/en-us/articles/360051931952-Routines-in-SmartThings>.
- [137] *Alexa Routines*. Accessed: April 2, 2026. Amazon. URL: <https://www.amazon.com/alexa-routines>.
- [138] G. Dorai, E. A. Williams, H. Chi, and R. A. Alo. "Is your Smart Home a Secure Home?" - Analysis of Smart Home Breaches and an Approach for Vulnerability Analysis and Device Isolation". In: *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. 2020, pp. 1–6. DOI: 10.1109/WF-IoT48130.2020.9221420.
- [139] F. K. Santoso and N. C. H. Vun. "Securing IoT for smart home system". In: *2015 International Symposium on Consumer Electronics (ISCE)*. 2015, pp. 1–2. DOI: 10.1109/ISCE.2015.7177843.
- [140] T. C. Guru. *Networking 101 for the Self-Hoster: Building Your Digital Fortress From the Ground Up*. Accessed: April 2, 2026. The CyberSec Guru. Jan. 2026. URL: <https://thecybersecguru.com/self-hosting/networking-101-home-network-setup-self-hosting>.

- [141] *Microsoft Defender*. Accessed: April 2, 2026. Microsoft. URL: <https://www.microsoft.com/microsoft-365/microsoft-defender-for-individuals>.
- [142] V. A. L. Sobral, N. Saoda, R. Shah, W. Wang, and B. Campbell. “RetroIoT: retrofitting internet of things deployments by hiding data in battery readings”. In: *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. MobiCom ’22. Sydney, NSW, Australia: Association for Computing Machinery, 2022, pp. 69–81. ISBN: 9781450391818. DOI: 10.1145/3495243.3560536.
- [143] Tasmota. *Open source firmware for ESP devices*. Accessed: October 4, 2025. 2025. URL: <https://staars.github.io/docs>.
- [144] D. Breitenbacher, I. Homoliak, Y. L. Aung, Y. Elovici, and N. O. Tippenhauer. “HADES-IoT: A Practical and Effective Host-Based Anomaly Detection System for IoT Devices (Extended Version)”. In: *IEEE Internet of Things Journal* 9.12 (2022), pp. 9640–9658. DOI: 10.1109/JIOT.2021.3135789.
- [145] M. Nobakht, V. Sivaraman, and R. Boreli. “A Host-Based Intrusion Detection and Mitigation Framework for Smart Home IoT Using OpenFlow”. In: *2016 11th International Conference on Availability, Reliability and Security (ARES)*. ACM ICPS, Aug. 2016, pp. 147–156. DOI: 10.1109/ARES.2016.64.
- [146] M. F. Elrawy, A. I. Awad, and H. F. Hamed. “Intrusion detection systems for IoT-based smart environments: a survey”. In: *Journal of Cloud Computing* 7.1 (2018), p. 21.
- [147] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici. “N-BaIoT—network-based detection of IoT botnet attacks using deep autoencoders”. In: *IEEE Pervasive Computing* 17.3 (2018), pp. 12–22.
- [148] U. H. Garba, A. N. Toosi, M. F. Pasha, and S. Khan. “SDN-based detection and mitigation of DDoS attacks on smart homes”. In: *Computer Communications* 221 (2024), pp. 29–41.
- [149] *Technicolor TG582n*. Accessed: April 2, 2026. OpenWrt. Nov. 2025. URL: <https://openwrt.org/toh/thomson/tg582n>.
- [150] *Technicolor TG789vac V2*. Accessed: April 2, 2026. OpenWrt. Apr. 2023. URL: https://openwrt.org/inbox/toh/technicolor/tg789vac_v2.
- [151] J. Habibi, D. Midi, A. Mudgerikar, and E. Bertino. “Heimdall: Mitigating the Internet of Insecure Things”. In: *IEEE Internet of Things Journal* 4.4 (2017), pp. 968–978.

- [152] A. Pashamokhtari, H. H. Gharakheili, and V. Sivaraman. "Progressive Monitoring of IoT Networks Using SDN and Cost-Effective Traffic Signatures". In: *2020 Workshop on Emerging Technologies for Security in IoT (ETSecIoT)*. IEEE, Apr. 2020, pp. 1–6. DOI: 10.1109/ETSecIoT50046.2020.00005.
- [153] M. Tavallaei, E. Bagheri, W. Lu, and A. A. Ghorbani. "A detailed analysis of the KDD CUP 99 data set". In: *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. 2009, pp. 1–6. DOI: 10.1109/CISDA.2009.5356528.
- [154] L. Metongnon, R. Sadre, and E. C. Ezin. "Distributed Middlebox Architecture for IoT Protection". In: *2019 15th International Conference on Network and Service Management (CNSM)*. ISSN: 2165-963X. Oct. 2019, pp. 1–7. DOI: 10.23919/CNSM46954.2019.9012738.
- [155] M. Hasan, M. M. Islam, M. I. I. Zarif, and M. M. A. Hashem. "Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches". In: *Internet of Things 7* (Sept. 2019), p. 100059. ISSN: 2542-6605. DOI: 10.1016/j.iot.2019.100059. (Visited on 10/18/2021).
- [156] C. Grajeda, F. Breitingner, and I. Baggili. "Availability of datasets for digital forensics—and what is missing". In: *Digital Investigation 22* (2017), S94–S105.
- [157] P. Penrose, R. Macfarlane, and W. J. Buchanan. "Approaches to the classification of high entropy file fragments". In: *Digital Investigation 10.4* (2013), pp. 372–384.
- [158] K. Kousias, M. Rajiullah, G. Caso, U. Ali, O. Alay, A. Brunstrom, L. De Nardis, M. Neri, and M.-G. Di Benedetto. "A Large-Scale Dataset of 4G, NB-IoT, and 5G Non-Standalone Network Measurements". In: *IEEE Communications Magazine* 62.5 (2024), pp. 44–49. DOI: 10.1109/MCOM.011.2200707.
- [159] G. K. Ndonga and R. Sadre. "Network trace generation for flow-based IDS evaluation in control and automation systems". In: *International Journal of Critical Infrastructure Protection* 31 (2020), p. 100385.
- [160] C. Archer. "What's the difference between SCADA and IoT?" Accessed: July 17, 2022. WellAware. Mar. 2021. URL: <https://blog.wellaware.us/blog/difference-between-scada-and-iot-scada-vs-iot>.
- [161] H. M. Song, J. Woo, and H. K. Kim. "In-vehicle network intrusion detection using deep convolutional neural network". In: *Vehicle Communications* 21 (2020), p. 100198.

- [162] A. L. Buczak and E. Guven. "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection". In: *IEEE Communications Surveys & Tutorials* 18.2 (2016), pp. 1153–1176. DOI: 10.1109/COMST.2015.2494502.
- [163] R. Rawassizadeh and D. Kotz. "Datasets for mobile, wearable and IoT research". In: *GetMobile: Mobile Computing and Communications* 20.4 (2017), pp. 5–7.
- [164] X. Jing, Z. Yan, and W. Pedrycz. "Security Data Collection and Data Analytics in the Internet: A Survey". In: *IEEE Communications Surveys & Tutorials* 21.1 (2019), pp. 586–618. DOI: 10.1109/COMST.2018.2863942.
- [165] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki. "Network Intrusion Detection for IoT Security Based on Learning Techniques". In: *IEEE Communications Surveys & Tutorials* 21.3 (2019), pp. 2671–2701. DOI: 10.1109/COMST.2019.2896380.
- [166] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho. "A survey of network-based intrusion detection data sets". In: *Computers & Security* 86 (2019), pp. 147–167.
- [167] M. Conti, D. Donadel, and F. Turrin. "A Survey on Industrial Control System Testbeds and Datasets for Security Research". In: *IEEE Communications Surveys & Tutorials* 23.4 (2021), pp. 2248–2294. DOI: 10.1109/COMST.2021.3094360.
- [168] P. M. S. Sánchez, J. M. J. Valero, A. H. Celdrán, G. Bovet, M. G. Pérez, and G. M. Pérez. "A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets". In: *IEEE Communications Surveys & Tutorials* 23.2 (2021), pp. 1048–1077. DOI: 10.1109/COMST.2021.3064259.
- [169] "1998 DARPA Intrusion Detection Evaluation Dataset". Accessed: February 02, 2022. MIT Lincoln Laboratory. URL: <https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset>.
- [170] "Data | KDD Cup 1999: Computer network intrusion detection". Accessed: January 12, 2022. KDD. URL: <https://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data>.
- [171] "NSL-KDD dataset". Accessed: January 27, 2022. Canadian Institute for Cybersecurity. URL: <https://www.unb.ca/cic/datasets/ns1.html>.
- [172] *IMC Conference*. Accessed: February 10, 2022. ACM Sigcomm. URL: <https://www.sigcomm.org/events/imc-conference> (visited on 02/10/2022).

- [173] *Passive and Active Measurement Conference 2022*. Accessed: February 10, 2022. URL: <https://pam2022.nl> (visited on 02/10/2022).
- [174] *TMA Conference*. Accessed: February 10, 2022. URL: <https://tma.ifip.org> (visited on 02/10/2022).
- [175] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. *Feature-oriented domain analysis (FODA) feasibility study*. Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.
- [176] P.-Y. Schobbens, P. Heymans, and J.-C. Trigaux. "Feature Diagrams: A Survey and a Formal Semantics". In: *14th IEEE International Requirements Engineering Conference (RE'06)*. 2006, pp. 139–148. DOI: 10.1109/RE.2006.23.
- [177] *"Riverbed Modeler | Discrete Event Simulator for Network Simulation"*. Accessed: January 11, 2022. Riverbed. URL: <https://www.riverbed.com/products/npm/riverbed-modeler.html>.
- [178] *The Network Simulator - ns-2*. Accessed: October 23, 2025. nsnam. URL: <https://www.isi.edu/websites/nsnam/ns>.
- [179] *"ns-3 | A discrete-event network simulator for internet systems"*. Accessed: January 11, 2022. URL: <https://www.nsnam.org/>.
- [180] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. "Cross-Level Sensor Network Simulation with COOJA". In: *Proceedings. 2006 31st IEEE conference on local computer networks*. IEEE. 2006, pp. 641–648.
- [181] L. Spitzner. *Honeypots: tracking hackers*. Vol. 1. Addison-Wesley Reading, 2003.
- [182] R. Bace and P. Mell. "Intrusion Detection Systems". In: *Special Publication (NIST SP)*. Gaithersburg, MD, United States: National Institute of Standards and Technology, 2001.
- [183] M. G. Gouda and A. X. Liu. "Structured firewall design". In: *Computer networks* 51.4 (2007), pp. 1106–1120.
- [184] S. J. Saidi, A. M. Mandalari, R. Kolcun, H. Haddadi, D. J. Dubois, D. Choffnes, G. Smaragdakis, and A. Feldmann. "A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild". In: *Proceedings of the ACM Internet Measurement Conference*. IMC '20. New York, NY, USA: ACM, Oct. 2020, pp. 87–100. ISBN: 978-1-4503-8138-3. DOI: 10.1145/3419394.3423650. (Visited on 10/28/2021).
- [185] H. Nissenbaum. "Privacy in context". In: *Privacy in Context*. Stanford University Press, 2009.

- [186] H. Kang, D. H. Ahn, G. M. Lee, J. D. Yoo, K. H. Park, and H. K. Kim. "IoT network intrusion dataset". DOI: <http://dx.doi.org/10.2127/q70p-q449>. IEEE Dataport. 2019. DOI: 10.21227/q70p-q449.
- [187] S. Dowling, M. Schukat, and H. Melvin. "A ZigBee honeypot to assess IoT cyberattack behaviour". In: *2017 28th Irish signals and systems conference (ISSC)*. IEEE. 2017, pp. 1–6.
- [188] "BLEBeacon Dataset". Accessed: February 3, 2022. URL: <https://github.com/dimisik/BLEBeacon-Dataset>.
- [189] M. Aernouts, R. Berkvens, K. Van Vlaenderen, and M. Weyn. "Sigfox and LoRaWAN Datasets for Fingerprint Localization in Large Urban and Rural Areas". Version 1.3. DOI: <http://dx.doi.org/10.5281/zenodo.3904158>. Zenodo. July 2019. DOI: 10.5281/zenodo.3904158.
- [190] L. Bhatia, M. Breza, R. Marfievici, and J. A. McCann. "LoED: The LoRaWAN at the Edge Dataset". DOI: <http://dx.doi.org/10.5281/zenodo.4121430>. Zenodo. Sept. 2020. DOI: 10.5281/zenodo.4121430.
- [191] "LoRa Radio Data". Accessed: January 26, 2022. InterDigital. URL: https://www.interdigital.com/data_sets/lora-radio-data.
- [192] A. Elmaghub and B. Hamdaoui. "Release Note: Comprehensive LoRa RF Datasets for Device Fingerprinting Using Deep Learning". Accessed: February 16, 2022. July 2021. URL: http://research.engr.oregonstate.edu/hamdaoui/sites/research.engr.oregonstate.edu.hamdaoui/files/release_note_2021.pdf.
- [193] D.-G. Akestoridis, M. Harishankar, M. Weber, and P. Tague. "CRAWDAD dataset cmu/zigbee-smarthome (v. 2020-05-26)". May 2020.
- [194] A. Collen, N. A. Nijdam, J. Augusto-Gonzalez, S. K. Katsikas, K. M. Giannoutakis, G. Spathoulas, E. Gelenbe, K. Votis, D. Tzovaras, N. Ghavami, et al. "GHOST-IoT-data-set | Smart-home network traffic IoT dataset". Accessed: January 24, 2022. URL: <https://github.com/gspathoulas/ghost-iot-dataset>.
- [195] D.-G. Akestoridis and P. Tague. "CRAWDAD dataset cmu/zigbee-eda (v. 2021-10-22)". DOI: <http://dx.doi.org/10.15783/t8mt-a674>. CRAWDAD. Oct. 2021. DOI: 10.15783/t8mt-a674.
- [196] M. Zolanvari, Z. Yang, K. Khan, R. Jain, and N. Meskin. "WUSTL-IIOT-2021 Dataset for IIoT Cybersecurity Research". Accessed: February 13, 2023. Oct. 2021. URL: <https://www.cse.wustl.edu/~jain/iiot2/index.html>.
- [197] J. Mathews, P. Chatterjee, and S. Banik. "CoAP-DDoS - DDoS Dataset for CoAP Protocol in IoT". Accessed: September 28, 2022. URL: <https://www.kaggle.com/datasets/jaredalanmathews/coapddos>.

- [198] D. Kotz, T. Henderson, I. Abyzov, and J. Yeo. "CRAWDAD dataset dartmouth/campus (v. 2009-09-09)". DOI: <http://dx.doi.org/10.15783/C7F59T>. CRAWDAD. Sept. 2009. DOI: 10.15783/C7F59T.
- [199] "The CAIDA "DDoS Attack 2007" Dataset". Accessed: January 1, 2022. CAIDA. URL: https://www.caida.org/catalog/datasets/ddos-20070804_dataset.
- [200] "Cyber Research Center - Data Sets". Accessed: February 1, 2022. United States Military Academy West Point. URL: <https://www.westpoint.edu/centers-and-research/cyber-research-center/data-sets>.
- [201] "DARPA_2009_DDoS_attack-20091105". Accessed: February 8, 2022. Impact Cyber Trust. 2009. URL: https://www.impactcybertrust.org/dataset_view?idDataset=742.
- [202] "WITS: Waikato Internet Traffic Storage". Accessed: January 28, 2022. WAND Network Research Group. URL: <https://wand.net.nz/wits>.
- [203] "Traffic Data from Kyoto University's Honeypots". Accessed: January 31, 2022. URL: http://www.takakura.com/Kyoto_data.
- [204] P. Gogoi, M. H. Bhuyan, D. Bhattacharyya, and J. K. Kalita. "Packet and Flow Based Network Intrusion Dataset". In: *International Conference on Contemporary Computing*. Springer. 2012, pp. 322–334.
- [205] "The CTU-13 Dataset. A Labeled Dataset with Botnet, Normal and Background traffic." Accessed: February 1, 2022. Stratosphere Lab. URL: <https://www.stratosphereips.org/datasets-ctu13>.
- [206] "The UNSW-NB15 Dataset". Accessed: January 26, 2022. UNSW Sydney. URL: <https://research.unsw.edu.au/projects/unsw-nb15-dataset>.
- [207] "H23Q". Accessed: February 7, 2023. University of the Aegean. URL: <https://icsdweb.aegean.gr/awid/other-datasets/H23Q>.
- [208] A. Sivanathan, D. Sherratt, H. H. Gharakheili, V. Sivaraman, and A. Vishwanath. "Low-cost flow-based security solutions for smart-home IoT devices". In: *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. IEEE. 2016, pp. 1–6.
- [209] "Urban_IoT_DDoS_Data". Accessed: February 11, 2022. ANRG USC. URL: https://github.com/ANRGUSC/Urban_IoT_DDoS_Data.
- [210] V. H. Bezerra, V. G. T. da Costa, R. A. Martins, S. B. Junior, R. S. Miani, and B. B. Zarpelão. "Providing IoT host-based datasets for intrusion detection research". In: *Proceedings of the 18th Brazilian Symposium on Information and Computational Systems Security*. Natal: SBC, 2018, pp. 15–28.

- [211] *"detection_of_IoT_botnet_attacks_N_BaIoT Data Set"*. Accessed: January 20, 2022. UCI Machine Learning Repository. URL: https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT.
- [212] R. Doshi, N. Apthorpe, and N. Feamster. "Machine learning DDoS detection for consumer Internet of Things devices". In: *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2018, pp. 29–35.
- [213] F.-X. Aubet and M.-O. Pahl. *"DS2OS traffic traces"*. Accessed: February 28, 2022. 2018. URL: <https://www.kaggle.com/francoisxa/ds2ostrafictraces>.
- [214] *"IoT Benign and Attack Traces"*. Accessed: January 1, 2022. UNSW Sydney. URL: <https://iotanalytics.unsw.edu.au/attack-data>.
- [215] *"The Bot-IoT Dataset"*. Accessed: January 19, 2022. UNSW Sydney. URL: <https://research.unsw.edu.au/projects/bot-iot-dataset>.
- [216] S. Garcia, A. Parmisano, and M. J. Erquiaga. *"IoT-23: A labeled dataset with malicious and benign IoT network traffic"*. Version 1.0.0. DOI: <http://dx.doi.org/10.5281/zenodo.4743746>. Zenodo. Jan. 2020. DOI: 10.5281/zenodo.4743746.
- [217] L. Vigoya, D. Fernandez, V. Carneiro, and F. Cacheda. *"DAD / DATASET for Anomaly Detection in IoT Networks"*. Accessed: January 24, 2022. URL: <https://github.com/dad-repository/dad>.
- [218] A. Guerra-Manzanares, J. Medina-Galindo, H. Bahsi, and S. Nömm. *"MedBIoT - Medium-sized IoT Botnet Data Set"*. Accessed: January 24, 2022. URL: <https://cs.taltech.ee/research/data/medbiot>.
- [219] Sarica, Alper Kaan. *"SDN-Dataset"*. Accessed: Feb 17, 2022. URL: <https://github.com/AlperKaan35/SDN-Dataset>.
- [220] *The TON_IoT Datasets*. Accessed: January 20, 2022. UNSW Sydney. URL: <https://research.unsw.edu.au/projects/toniot-datasets>.
- [221] Z. Liu, N. Thapa, A. Shaver, K. Roy, M. Siddula, X. Yuan, and A. Yu. "Using Embedded Feature Selection and CNN for Classification on CCD-INID-V1—A New IoT Dataset". In: *Sensors* 21.14 (2021), p. 4834.
- [222] M. Ahmed, S. Byreddy, A. Nutakki, L. F. Sikos, and P. Haskell-Dowland. *"ECU-IoHT"*. Accessed: January 24, 2022. URL: <https://ro.ecu.edu.au/datasets/48>.
- [223] I. Vaccari, G. Chiola, M. Aiello, M. Mongelli, and E. Cambiaso. *"MQTTset - A new dataset for machine learning techniques on MQTT"*. Accessed: June 22, 2022. URL: <https://www.kaggle.com/datasets/cnr-ieiit/mqttset>.

- [224] H. Gandhi, M. Mehra, and V. Ribeiro. "BOND: Efficient and Frugal DL Model Co-Design for Botnet Detection on IoT Gateways". In: *The First International Conference on AI-ML-Systems*. AIMLSystems 2021. Bangalore, India: Association for Computing Machinery, 2021. ISBN: 9781450385947. DOI: 10.1145/3486001.3486237.
- [225] M. Ahmed, D. Cox, B. Simpson, and A. Aloufi. "ECU-IoFT: A Dataset for Analysing Cyber-Attacks on Internet of Flying Things". Accessed: June 22, 2022. URL: <https://github.com/iMohi/ECU-IoFT>.
- [226] S. Suthaharan, M. Alzahrani, S. Rajasegarar, C. Leckie, and M. Palaniswami. "Labelled Wireless Sensor Network Data Repository (LWS-NDR)". Accessed: January 27, 2022. URL: <https://home.uncg.edu/cmp/downloads/lwsndr.html>.
- [227] "WSN-DS". Accessed: February 3, 2022. Security Engineering Lab. URL: https://sel.psu.edu.sa/Research/datasets/2016_WSN-DS.php.
- [228] A. Verma and V. Ranga. "Evaluation of network intrusion detection systems for RPL based 6LoWPAN networks in IoT". In: *Wireless Personal Communications* 108.3 (2019), pp. 1571–1594.
- [229] Y. Al-Hadhrami and F. K. Hussain. "Real time dataset generation framework for intrusion detection systems in IoT". In: *Future Generation Computer Systems* 108 (2020), pp. 414–423. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2020.02.051>.
- [230] F. Sadikin, T. van Deursen, and S. Kumar. "A ZigBee Intrusion Detection System for IoT using Secure and Efficient Data Collection". In: *Internet of Things* 12 (2020), p. 100306.
- [231] Y.-W. Chen, J.-P. Sheu, Y.-C. Kuo, and N. Van Cuong. "Design and implementation of IoT DDoS attacks detection system based on machine learning". In: *2020 European Conference on Networks and Communications (EuCNC)*. IEEE. 2020, pp. 122–127.
- [232] A. Lacava, E. Giacomini, F. D'Alterio, and F. Cuomo. "Intrusion Detection System for Bluetooth Mesh Networks: Data Gathering and Experimental Evaluations". In: *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE. 2021, pp. 661–666.
- [233] Kamaldeep, M. Malik, M. Dutta, and J. Granjal. "IoT-Sentry: A Cross-Layer-Based Intrusion Detection System in Standardized Internet of Things". In: *IEEE Sensors Journal* 21.24 (2021), pp. 28066–28076. DOI: 10.1109/JSEN.2021.3124886.

- [234] M. Al-Hawawreh, E. Sitnikova, and N. Aboutorab. *X-IIoTID: A Connectivity- and Device-agnostic Intrusion Dataset for Industrial Internet of Things*. 2021. DOI: 10.21227/mpb6-py55. URL: <https://dx.doi.org/10.21227/mpb6-py55>.
- [235] M. A. Ferrag. *"Edge-IIoTset Cyber Security Dataset of IoT & IIoT"*. Accessed: February 13, 2023. 2022. URL: <https://www.kaggle.com/datasets/mohamedamineferrag/edgeiiotset-cyber-security-dataset-of-iiot-iiot>.
- [236] A. Vasudevan, E. Harshini, and S. Selvakumar. "SSENet-2011: A Network Intrusion Detection System dataset and its comparison with KDD CUP 99 dataset". In: *2011 Second Asian Himalayas International Conference on Internet (AH-ICI)*. 2011, pp. 1–5. DOI: 10.1109/AHICI.2011.6113948.
- [237] *"ADFA IDS Datasets"*. Accessed: February 3, 2022. UNSW Sydney. URL: <https://research.unsw.edu.au/projects/adfa-ids-datasets>.
- [238] *"Intrusion detection evaluation dataset (ISCXIDS2012)"*. Accessed: February 9, 2022. Canadian Institute for Cybersecurity. URL: <https://www.unb.ca/cic/datasets/ids.html>.
- [239] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón. *"UGR'16: A New Dataset for the Evaluation of Cyclostationarity-Based Network IDSs"*. Accessed: January 24, 2022. URL: <https://nesg.ugr.es/nesg-ugr16>.
- [240] *"CIC-IDS-2017"*. Accessed: January 24, 2022. Canadian Institute for Cybersecurity. 2017. URL: <https://www.unb.ca/cic/datasets/ids-2017.html>.
- [241] *"CSE-CIC-IDS2018"*. Accessed: January 24, 2022. Canadian Institute for Cybersecurity. 2018. URL: <https://www.unb.ca/cic/datasets/ids-2018.html>.
- [242] *"CICDDoS2019"*. Accessed: January 24, 2022. Canadian Institute for Cybersecurity. 2019. URL: <https://www.unb.ca/cic/datasets/ddos-2019.html>.
- [243] *"CIRA-CIC-DoHBrw-2020"*. Accessed: February 4, 2022. Canadian Institute for Cybersecurity. 2020. URL: <https://www.unb.ca/cic/datasets/dohbrw-2020.html>.
- [244] R. Damasevicius, A. Venckauskas, S. Grigaliunas, J. Toldinas, N. Morkevicius, T. Aleliunas, and P. Smuikys. *"LITNET-2020: an annotated real-world network flows dataset for network intrusion detection"*. Accessed: January 24, 2022. URL: <https://dataset.litnet.lt>.

- [245] "Machine Learning-Based NIDS Datasets". Accessed: March 30, 2022. The University of Queensland. URL: https://staff.itee.uq.edu.au/marius/NIDS_datasets.
- [246] "AWID2". Accessed: January 25, 2022. University of the Aegean. URL: <https://icsdweb.aegean.gr/awid/awid2>.
- [247] S. Fu and Y. Zhang. "The due/package-delivery dataset (v. 2015-04-01)". Accessed: March 20, 2023. 2015. URL: <https://ieee-dataport.org/open-access/crawdad-duepacket-delivery>.
- [248] "AWID3". Accessed: January 25, 2022. University of the Aegean. URL: <https://icsdweb.aegean.gr/awid/awid3>.
- [249] S. Marchal. "IoT devices captures". Accessed: February 8, 2022. Aalto University. Apr. 2017. URL: <https://research.aalto.fi/en/datasets/iot-devices-captures>.
- [250] "IoT Traffic Traces". Accessed: January 12, 2022. UNSW Sydney. URL: <https://iotanalytics.unsw.edu.au/iottraces.html>.
- [251] "MUD Profiles". Accessed: January 12, 2022. UNSW Sydney. URL: <http://s://iotanalytics.unsw.edu.au/mudprofiles>.
- [252] "IoT Detection in the Wild (IMC '20)". Accessed: February 2, 2022. Mon(IoT)r Research Group. URL: <https://moniotrlab.ccis.neu.edu/imc20>.
- [253] "IoT IPFIX Records". Accessed: January 27, 2022. UNSW Sydney. URL: <https://iotanalytics.unsw.edu.au/iotipfixrecords>.
- [254] "Data and code for TMA 2021 paper "Revisiting IoT Device Identification"". Accessed: February 28, 2022. URL: <https://github.com/DADABox/revisiting-iot-device-identification>.
- [255] B. Purnama, Sharipuddin, Kurniabudi, R. Budiarto, D. Stiawan, and D. Hanapi. "Monitoring Connectivity of Internet of Things Device on Zigbee Protocol". In: *2018 International Conference on Electrical Engineering and Computer Science (ICECOS)*. 2018, pp. 351–356. DOI: 10.1109/ICECOS.2018.8605225.
- [256] "IoT Information Exposure (IMC '19)". Accessed: January 20, 2022. Mon(IoT)r Research Group. URL: <https://moniotrlab.ccis.neu.edu/imc19>.
- [257] L. Barman, A. Dumur, A. Pyrgelis, and J.-P. Hubaux. "Repository for the paper "Every Byte Matters: Traffic Analysis of Bluetooth Wearable Devices"". Accessed: January 25, 2022. 2021. URL: https://github.com/ldsec/every_byte_matters.

- [258] M. Aernouts, R. Berkvens, K. Van Vlaenderen, and M. Weyn. “Sigfox and LoRaWAN datasets for fingerprint localization in large urban and rural areas”. In: *Data* 3.2 (2018), p. 13.
- [259] A. Al-Shawabka, P. Pietraski, S. B. Pattar, F. Restuccia, and T. Melodia. “DeepLoRa: Fingerprinting LoRa Devices at Scale Through Deep Learning and Data Augmentation”. In: *Proceedings of the Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*. 2021, pp. 251–260.
- [260] F. Carrino, A. Janka, O. Abou Khaled, and E. Mugellini. “LoRaLoc: Machine learning-based fingerprinting for outdoor geolocation using LoRa”. In: *2019 6th Swiss Conference on Data Science (SDS)*. IEEE. 2019, pp. 82–86.
- [261] J. Purohit, X. Wang, S. Mao, X. Sun, and C. Yang. “Fingerprinting-based indoor and outdoor localization with LoRa and deep learning”. In: *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE. 2020, pp. 1–6.
- [262] A. Elmaghub and B. Hamdaoui. “LoRa device fingerprinting in the wild: Disclosing RF data-driven fingerprint sensitivity to deployment variability”. In: *IEEE Access* 9 (2021), pp. 142893–142909.
- [263] I. Zavalysyn. “Building Private-by-Design IoT Systems”. PhD thesis. NOVA University, Portugal 2021.
- [264] P. M. Chanal and M. S. Kakkasageri. “Security and privacy in IOT: a survey”. In: *Wireless Personal Communications* 115 (2020), pp. 1667–1693.
- [265] C. Perera, R. Ranjan, L. Wang, S. U. Khan, and A. Y. Zomaya. “Big data privacy in the internet of things era”. In: *IT Professional* 17.3 (2015), pp. 32–39.
- [266] M. Cermák, T. Jirsík, P. Velan, J. Komárková, S. Spacek, M. Drasar, and T. Plesnik. “Towards Provable Network Traffic Measurement and Analysis via Semi-Labeled Trace Datasets”. In: *Network Traffic Measurement and Analysis Conference, TMA 2018, Vienna, Austria, June 26-29, 2018*. IEEE, 2018, pp. 1–8. DOI: 10.23919/TMA.2018.8506498.
- [267] R. Pang, M. Allman, V. Paxson, and J. Lee. “The devil and packet trace anonymization”. In: *ACM SIGCOMM Computer Communication Review* 36.1 (2006), pp. 29–38.

- [268] M. Mohammady, L. Wang, Y. Hong, H. Louafi, M. Pourzandi, and M. Debbabi. "Preserving both privacy and utility in network trace anonymization". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 459–474.
- [269] M. Burkhart, D. Schatzmann, B. Trammell, E. Boschi, and B. Platner. "The role of network trace anonymization under attack". In: *ACM SIGCOMM Computer Communication Review* 40.1 (2010), pp. 5–11.
- [270] A. B. and Alberto Dainotti and A. Pescapè. "A tool for the generation of realistic network workload for emerging networking scenarios". In: *Computer Networks* 56.15 (2012), pp. 3531–3547.
- [271] "D-ITG, Distributed Internet Traffic Generator". Accessed: February 15, 2022. URL: <http://traffic.comics.unina.it/software/ITG>.
- [272] J. Sommers, H. Kim, and P. Barford. "Harpoon: a flow-level traffic generator for router and network tests". In: *ACM SIGMETRICS Performance Evaluation Review* 32.1 (2004), pp. 392–392.
- [273] J. Sommers. "Harpoon: A Flow-level Traffic Generator". Accessed: February 15, 2022. Dec. 2005. URL: <https://jsommers.github.io/harpoon>.
- [274] xorrsprsp. "fudp - threaded udp flooder". Accessed: February 15, 2022. Nov. 2017. URL: <https://github.com/xorrsprsp/fudp>.
- [275] "Trex". Accessed: September 26, 2022. Cisco. 2019. URL: <https://trex-tgn.cisco.com>.
- [276] S. Ghazanfar, F. Hussain, A. U. Rehman, U. U. Fayyaz, F. Shahzad, and G. A. Shah. "IoT-Flock: An Open-source Framework for IoT Traffic Generation". In: *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*. 2020, pp. 1–6. DOI: 10.1109/ICETST49965.2020.9080732.
- [277] "IoT-Flock - An Open-Source Framework For IoT Traffic Generation". Accessed: June 24, 2022. URL: <https://github.com/ThingzDefense/IoT-Flock>.
- [278] W.-H. Hsu, Q. Li, X.-H. Han, and C.-W. Huang. "A hybrid IoT traffic generator for mobile network performance assessment". In: *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2017, pp. 441–445. DOI: 10.1109/IWCMC.2017.7986326.
- [279] S. Hui, H. Wang, Z. Wang, X. Yang, Z. Liu, D. Jin, and Y. Li. "Knowledge Enhanced GAN for IoT Traffic Generation". In: *Proceedings of the ACM Web Conference 2022*. 2022, pp. 3336–3346.

- [280] H. Nguyen-An, T. Silverston, T. Yamazaki, and T. Miyoshi. "Generating IoT Traffic in Smart Home Environment". In: *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*. 2020, pp. 1–2. DOI: 10.1109/CCNC46108.2020.9045343.
- [281] A. Lavric, A. I. Petrariu, E. Coca, and V. Popa. "LoRa Traffic Generator Based on Software Defined Radio Technology for LoRa Modulation Orthogonality Analysis: Empirical and Experimental Evaluation". In: *Sensors* 20.15 (2020). ISSN: 1424-8220.
- [282] M. Gucciardo, I. Tinnirello, and D. Garlisi. "A cell-level traffic generator for LoRa networks". In: *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. 2017, pp. 474–476.
- [283] J. Franco, A. Aris, B. Canberk, and A. S. Uluagac. "A Survey of Honey-pots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems". In: *IEEE Communications Surveys & Tutorials* 23.4 (2021), pp. 2351–2383. DOI: 10.1109/COMST.2021.3106669.
- [284] *HS110 | Kasa Smart Wi-Fi Plug with Energy Monitoring*. Accessed: December 5, 2024. TP-Link. 2024. URL: <https://www.tp-link.com/en/home-networking/smart-plug/hs110>.
- [285] *Tapo P110 | Mini Smart Wi-Fi Socket, Energy Monitoring*. Accessed: December 5, 2024. TP-Link. 2024. URL: <https://www.tp-link.com/en/home-networking/smart-plug/tapo-p110/v1>.
- [286] *Woox R5024*. Accessed: December 5, 2024. Woox. 2024. URL: <https://wooxhome.com/products-c10/power-c2/woox-r5024-smart-plug-eu-p8>.
- [287] *Smart Plug (2019)*. Accessed: December 5, 2024. Samsung. 2024. URL: <https://www.samsung.com/uk/smartthings/outlet/smartthings-smart-plug-gp-wou019bbdwg>.
- [288] *Tapo L530E | Smart Wi-Fi Light Bulb, Multicolor*. Accessed: December 5, 2024. TP-Link. 2024. URL: <https://www.tp-link.com/en/home-networking/smart-bulb/tapo-l530e>.
- [289] *Alecto SMART-BULB10 DUO - Smart LED colour lamp with Wi-Fi, 2 pack*. Accessed: December 5, 2024. Alecto Home. 2024. URL: <https://alectohome.com/products/alecto-smart-bulb10-duo-smart-wifi-led-lamp-2-pack-wit>.
- [290] *Hue White and Color Ambiance A60 - E27 smart bulb - 800*. <https://www.philips-hue.com/en-my/p/hue-white-and-color-ambiance-a60---e27-smart-bulb---800/8718696725580>. Accessed: August 24, 2023. Philips, Aug. 2023.

- [291] *Tapo C200 | Pan/Tilt Home Security Wi-Fi Camera*. Accessed: December 5, 2024. TP-Link. 2024. URL: <https://www.tp-link.com/en/home-networking/cloud-camera/tapo-c200>.
- [292] *Mi Home Security Camera 360°*. Accessed: December 5, 2024. Xiaomi. 2024. URL: <https://www.mi.com/global/camera-360>.
- [293] *DCS-8000LH Mini HD WiFi Camera*. Accessed: December 5, 2024. D-Link. 2024. URL: <https://www.dlink.com/uk/en/products/dcs-8000lh-mini-hd-wifi-camera>.
- [294] *Samsung SmartThings Multipurpose Sensor*. <https://www.samsung.com/us/smart-home/smartthings/sensors/samsung-smartthings-multipurpose-sensor-f-mlt-us-2>. Accessed: August 24, 2023. Samsung, Aug. 2023.
- [295] *Samsung SmartThings Presence Sensor*. <https://www.samsung.com/us/support/smart-home/smartthings/sensors/presence-sensor>. Accessed: February 26, 2026. Samsung, Oct. 2023. URL: <https://www.samsung.com/us/support/smart-home/smartthings/sensors/presence-sensor>.
- [296] *Hue Bridge*. Accessed: December 5, 2024. Philips. 2023. URL: <https://www.philips-hue.com/en-us/p/hue-bridge/046677458478>.
- [297] *Smart Home Hub*. Accessed: December 5, 2024. Aeotec. 2024. URL: <https://aeotec.com/products/aeotec-smartthings-hub>.
- [298] *Amazon Echo Dot (2nd Generation) review*. <https://www.techradar.com/reviews/amazon-echo-dot-2nd-gen>. Accessed: February 26, 2026. Techradar. Nov. 2021. URL: <https://www.techradar.com/reviews/amazon-echo-dot-2nd-gen>.
- [299] *Do the SmartThings! Control your home with ease*. Accessed: December 2, 2024. Samsung. 2024. URL: <https://www.samsung.com/us/smartthings>.
- [300] *The Alexa App*. Accessed: September 14, 2025. Amazon. 2025. URL: <https://www.amazon.com/Alexa-App/b?node=18354642011>.
- [301] *Kasa Smart*. Accessed: March 21, 2025. TP-Link Systems Inc. 2025. URL: https://play.google.com/store/apps/details?id=com.tplink.kasa_android.
- [302] *TP-Link Tapo*. Accessed: March 21, 2025. TP-Link Systems Inc. 2025. URL: <https://play.google.com/store/apps/details?id=com.tplink.iot>.
- [303] *Tuya - Smart Life, Smart Living*. Accessed: September 14, 2025. Tuya Smart Inc. 2025. URL: <https://play.google.com/store/apps/details?id=com.tuya.smart>.

- [304] *The Philips Hue app*. Accessed: September 14, 2025. Philips Hue. 2025. URL: <https://www.philips-hue.com/en-us/explore-hue/apps/bridge>.
- [305] *Hue Essentials*. Accessed: March 21, 2025. SmartFusionLabs. 2025. URL: <https://play.google.com/store/apps/details?id=com.superthomaslab.hueessentials>.
- [306] *Xiaomi Home*. Accessed: September 14, 2025. Beijing Xiaomi Mobile Software Co.,Ltd. 2025. URL: <https://play.google.com/store/apps/details?id=com.xiaomi.smarthome>.
- [307] *The mydlink™ smart home app*. Accessed: September 14, 2025. D-Link Corporation. 2025. URL: <https://www.dlink.com/us/en/for-home/mydlink-app>.
- [308] *TP-Link TL-WDR4900 (N900)*. Accessed: November 24, 2025. OpenWrt. 2025. URL: <https://openwrt.org/toh/tp-link/tl-wdr4900>.
- [309] *Linksys WRT1200AC*. Accessed: March 21, 2025. OpenWrt. 2025. URL: <https://openwrt.org/toh/linksys/wrt1200ac>.
- [310] *CORE-X4: professional, efficient and practical smartphone*. Accessed: December 5, 2024. Crosscall. 2024. URL: https://www.crosscall.com/en_GB/core-x4-COX4.MASTER.html.
- [311] *Pixel 9a*. Accessed: July 17, 2025. Google. July 2025. URL: https://store.google.com/product/pixel_9a.
- [312] *Android Debug Bridge (adb)*. Accessed: July 15, 2025. Google. July 2025. URL: <https://developer.android.com/tools/adb>.
- [313] *Maestro Documentation*. Accessed: October 3, 2025. Maestro. Sept. 2025. URL: <https://maestro.mobile.dev>.
- [314] A. M. Mandalari, D. J. Dubois, R. Kolcun, M. T. Paracha, H. Haddadi, and D. Choffnes. “Blocking Without Breaking: Identification and Mitigation of Non-Essential IoT Traffic”. In: *Proceedings on Privacy Enhancing Technologies*. 2021. (Visited on 06/20/2024).
- [315] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. “Image Quality Assessment: From Error Visibility to Structural Similarity”. In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.
- [316] *python-kasa*. Accessed: October 1, 2025. python-kasa developers. Oct. 2025. URL: <https://python-kasa.readthedocs.io/en/stable>.
- [317] *tapo 0.8.6*. Accessed: October 1, 2025. Python Software Foundation. Oct. 2025. URL: <https://pypi.org/project/tapo>.

- [318] *pysmartthings 3.3.0*. Accessed: October 1, 2025. Python Software Foundation. Oct. 2025. URL: <https://pypi.org/project/pysmartthings>.
- [319] *aiohue 4.8.0*. Accessed: October 1, 2025. Python Software Foundation. Oct. 2025. URL: <https://pypi.org/project/aiohue/>.
- [320] E. Lear, R. Droms, and D. Romascanu. *Manufacturer Usage Description Specification*. RFC 8520. RFC Editor, Mar. 2019. 60 pp. DOI: 10.17487/RFC8520.
- [321] I. Martins, J. S. Resende, P. R. Sousa, S. Silva, L. Antunes, and J. Gama. “Host-based IDS: A review and open issues of an anomaly detection system in IoT”. In: *Future Generation Computer Systems* 133 (2022), pp. 95–113.
- [322] C. Fu, Q. Zeng, and X. Du. “HAWatcher: Semantics-Aware Anomaly Detection for Appified Smart Homes”. In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX, Aug. 2021, pp. 4223–4240. ISBN: 978-1-939133-24-3.
- [323] E. Lear and O. Friel. *Bandwidth Profiling Extensions for MUD*. Tech. rep. draft-lear-opsawg-mud-bw-profile-01. Backup Publisher: Internet Engineering Task Force Num Pages: 9. IETF, July 2019.
- [324] S. N. Matheu, J. L. Hernández-Ramos, S. Pérez, and A. F. Skarmeta. “Extending MUD Profiles Through an Automated IoT Security Testing Methodology”. In: *IEEE Access* 7 (2019), pp. 149444–149463. DOI: 10.1109/ACCESS.2019.2947157.
- [325] N. Mazhar, R. Salleh, M. Zeeshan, and M. M. Hameed. “Role of Device Identification and Manufacturer Usage Description in IoT Security: A Survey”. In: *IEEE Access* 9 (2021), pp. 41757–41786. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3065123. (Visited on 01/04/2024).
- [326] E. Ronen and A. Shamir. “Extended Functionality Attacks on IoT Devices: The Case of Smart Lights”. In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, Mar. 2016, pp. 3–12. DOI: 10.1109/EuroSP.2016.13.
- [327] D. Uroz and R. J. Rodríguez. “Characterization and Evaluation of IoT Protocols for Data Exfiltration”. In: *IEEE Internet of Things Journal* 9.19 (Oct. 2022), pp. 19062–19072. ISSN: 2327-4662. DOI: 10.1109/JIOT.2022.3163469.
- [328] M. Jethanandani, S. Agarwal, L. Huang, and D. Blair. *YANG Data Model for Network Access Control Lists (ACLs)*. RFC 8519. RFC Editor, Mar. 2019. 60 pp. DOI: 10.17487/RFC8519.

- [329] T. Reddy, K. D. Wing, and B. Anderson. *Manufacturer Usage Description (MUD) for TLS and DTLS Profiles for Internet of Things (IoT) Devices*. RFC 9761. IETF, Apr. 2025. 33 pp. DOI: 10.17487/RFC9761.
- [330] I. Zavalysyn, N. Santos, R. Sadre, and A. Legay. “My House, My Rules: A Private-by-Design Smart Home Platform”. In: *MobiQuitous 2020 - 17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. MobiQuitous '20. Darmstadt, Germany: Association for Computing Machinery, 2021, pp. 273–282. ISBN: 9781450388405. DOI: 10.1145/3448891.3450333.
- [331] S. Guo, M. Lyu, and H. H. Gharakheili. *Realizing Open and Decentralized Marketplace for Exchanging Data of Expected IoT Behaviors*. <http://arxiv.org/abs/2401.00141>. 2023. arXiv: 2401.00141 [cs.CR].
- [332] *Netfilter hooks*. NFTables Wiki, Mar. 2023.
- [333] *The netfilter.org "libnetfilter_queue" project*. Netfilter, Aug. 2023.
- [334] *Jinja*. The Pallets Project, 2023.
- [335] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi. “HomeSnitch: behavior transparency and control for smart home IoT devices”. In: *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*. New York, NY, USA: ACM, May 2019, pp. 128–138. ISBN: 978-1-4503-6726-4. DOI: 10.1145/3317549.3323409. (Visited on 10/25/2021).
- [336] W. W. Daniel. *Applied Nonparametric Statistics*. Cengage Learning, 2000.
- [337] Z. Bobbitt. *Kruskal-Wallis Test: Definition, Formula, and Example*. Accessed: April 7, 2026. Statology. Mar. 2022. URL: <https://www.statology.org/kruskal-wallis-test>.
- [338] R. A. Fisher. *Statistical Methods for Research Workers (14th Edition)*. Oliver and Boyd, 1970.
- [339] Z. Bobbitt. *An Explanation of P-Values and Statistical Significance*. Accessed: April 8, 2026. Statology. Apr. 2020. URL: <https://www.statology.org/p-values-statistical-significance>.
- [340] M. Lin, H. C. Lucas, and G. Shmueli. “Too big to fail: Large samples and the p-value problem”. In: *Information Systems Research* 24.4 (2013), pp. 906–917. DOI: 10.1287/isre.2013.0480.
- [341] T. L. Kelley. “An unbiased correlation ratio measure”. In: *Proceedings of the National Academy of Sciences* 21.9 (1935), pp. 554–559.

- [342] M. Tomczak and E. Tomczak. "The need to report effect size estimates revisited. An overview of some recommended measures of effect size". In: (2014).
- [343] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences (2nd Edition)*. Routledge, 1988.
- [344] J. Khalilzadeh and A. D. Tasci. "Large sample size, significance level, and the effect size: Solutions to perils of using big data for academic research". In: *Tourism Management* 62 (2017), pp. 89–96. ISSN: 0261-5177. DOI: <https://doi.org/10.1016/j.tourman.2017.03.026>.
- [345] A. Hamza, H. H. Gharakheili, and V. Sivaraman. "Combining MUD Policies with SDN for IoT Intrusion Detection". In: *Proceedings of the 2018 Workshop on IoT Security and Privacy*. New York, NY, USA: ACM, Aug. 2018, pp. 1–7. ISBN: 978-1-4503-5905-4. DOI: 10.1145/3229565.3229571. (Visited on 10/19/2021).
- [346] T. Hu, D. J. Dubois, and D. Choffnes. "BehavIoT: Measuring Smart Home IoT Behavior Using Network-Inferred Behavior Models". In: *Proceedings of the 2023 ACM on Internet Measurement Conference*. Montreal QC Canada: ACM, Oct. 2023, pp. 421–436. ISBN: 9798400703829. DOI: 10.1145/3618257.3624829. (Visited on 10/26/2023).
- [347] *Home Assistant*. Accessed: February 6, 2026. URL: <https://www.home-assistant.io>.
- [348] *openHAB*. Accessed: February 6, 2026. openHAB Foundation e.V. 2026. URL: <https://www.openhab.org>.
- [349] M. Yahyazadeh, P. Podder, E. Hoque, and O. Chowdhury. "Ex-pat: Expectation-based Policy Analysis and Enforcement for Appified Smart-Home Platforms". In: *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*. SACMAT '19. Toronto ON, Canada: Association for Computing Machinery, 2019, pp. 61–72. ISBN: 9781450367530. DOI: 10.1145/3322431.3325107.
- [350] A. K. Sikder, L. Babun, Z. B. Celik, H. Aksu, P. McDaniel, E. Kirda, and A. S. Uluagac. "Who's Controlling My Device? Multi-User Multi-Device-Aware Access Control System for Shared Smart Home Environment". In: *ACM Trans. Internet Things* 3.4 (Sept. 2022). DOI: 10.1145/3543513.
- [351] M. Di Pierro. "What Is the Blockchain?" In: *Computing in Science & Engineering* 19.5 (2017), pp. 92–95. DOI: 10.1109/MCSE.2017.3421554.

- [352] H. Liu, D. Han, and D. Li. "Fabric-iot: A Blockchain-Based Access Control System in IoT". In: *IEEE Access* 8 (2020), pp. 18207–18218. doi: 10.1109/ACCESS.2020.2968492.
- [353] C. K. Rath, A. K. Mandal, and A. Sarkar. "Blockchain-based dynamic MUD profiles for tamper-proof IoT access control". In: *Journal of Information Security and Applications* 95 (2025), p. 104273. issn: 2214-2126. doi: <https://doi.org/10.1016/j.jisa.2025.104273>.
- [354] T. Reddy, D. Wing, and B. Anderson. *MUD (D)TLS profiles for IoT devices*. Internet-Draft draft-reddy-opsawg-mud-tls-12. Backup Publisher: Internet Engineering Task Force Num Pages: 34. IETF, Apr. 2023.
- [355] M. Laurent. "Extending a dynamic IoT firewall tool : Integration of TLS and MQTT protocols for smart home security". Prom.: Ramin Sadre. MA thesis. UCLouvain, 2025.
- [356] S. Morais and C. Farias. "INXU - A Security Extension for RFC 8520 to Give Fast Response to New Vulnerabilities on Domestic IoT Networks". In: *Proceedings of the 7th Workshop Pré-IETF*. ISSN: 2595-6388 event-place: Evento Online. Porto Alegre, RS, Brasil: SBC, pp. 1–14. doi: 10.5753/wpietf.2020.13792.
- [357] S. N. Matheu, A. Robles Enciso, A. Molina Zarca, D. Garcia-Carrillo, J. L. Hernández-Ramos, J. Bernal Bernabe, and A. F. Skarmeta. "Security Architecture for Defining and Enforcing Security Profiles in DLT/SDN-Based IoT Systems". In: *Sensors* 20.7 (Jan. 2020). Number: 7 Publisher: Multidisciplinary Digital Publishing Institute, p. 1882. doi: 10.3390/s20071882. (Visited on 11/04/2021).
- [358] A. M. Zarca, J. B. Bernabe, R. Trapero, D. Rivera, J. Villalobos, A. Skarmeta, S. Bianchi, A. Zafeiropoulos, and P. Gouvas. "Security management architecture for NFV/SDN-aware IoT systems". In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 8005–8020.
- [359] A. Gallez and Z. Zhang. "Extending a Smart Home firewall with complex interactions and network pattern detection". Prom.: Ramin Sadre. MA thesis. UCLouvain, 2025.
- [360] T. Swart. *Cron Jobs: The Complete Guide for 2026*. <https://cronitor.io/guides/cron-jobs>. Accessed: February 21, 2026. Cronitor. Jan. 2025. URL: <https://cronitor.io/guides/cron-jobs>.
- [361] G. Chu, N. Apthorpe, and N. Feamster. "Security and Privacy Analyses of Internet of Things Children's Toys". In: *IEEE Internet of Things Journal* 6.1 (2019), pp. 978–985. doi: 10.1109/JIOT.2018.2866423.

- [362] S. N. Matheu, J. L. Hernández-Ramos, S. Pérez, and A. F. Skarmeta. “Extending MUD Profiles Through an Automated IoT Security Testing Methodology”. In: *IEEE Access* 7 (2019), pp. 149444–149463. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2947157.
- [363] S. Singh, A. Atrey, M. L. Sichitiu, and Y. Viniotis. “Clearer than Mud: Extending Manufacturer Usage Description (MUD) for Securing IoT Systems”. In: *Internet of Things – ICIOT 2019*. Ed. by V. Issarny, B. Palanisamy, and L.-J. Zhang. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 43–57. ISBN: 978-3-030-23357-0. DOI: 10.1007/978-3-030-23357-0_4.
- [364] A. Hamza, D. Ranathunga, H. H. Gharakheili, M. Roughan, and V. Sivaraman. “Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles”. In: *Proceedings of the 2018 Workshop on IoT Security and Privacy*. New York, NY, USA: ACM, Aug. 2018, pp. 8–14. ISBN: 978-1-4503-5905-4. DOI: 10.1145/3229565.3229566. (Visited on 11/29/2021).
- [365] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi. “HomeSnitch: behavior transparency and control for smart home IoT devices”. In: *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec ’19. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 128–138. ISBN: 978-1-4503-6726-4. DOI: 10.1145/3317549.3323409. (Visited on 10/25/2021).
- [366] M. H. Mazhar and Z. Shafiq. “Characterizing Smart Home IoT Traffic in the Wild”. In: *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 2020, pp. 203–215. DOI: 10.1109/IoTDI49375.2020.00027.
- [367] Y. Wan, K. Xu, F. Wang, and G. Xue. “IoT Athena: Unveiling IoT Device Activities From Network Traffic”. In: *IEEE Transactions on Wireless Communications* 21.1 (Jan. 2022). Conference Name: IEEE Transactions on Wireless Communications, pp. 651–664. ISSN: 1558-2248. DOI: 10.1109/TWC.2021.3098608. (Visited on 10/23/2023).
- [368] *Zigbee | Complete IoT Solution*. Accessed: December 6, 2024. Connectivity Standards Alliance. 2024. URL: <https://csa-iot.org/all-solutions/zigbee>.
- [369] P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu. “Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards”. In: *Computer Communications* 30.7 (2007). *Wired/Wireless Internet Communications*, pp. 1655–1695.

- ISSN: 0140-3664. DOI: <https://doi.org/10.1016/j.comcom.2006.12.020>.
- [370] C. W. Badenhop, S. R. Graham, B. W. Ramsey, B. E. Mullins, and L. O. Mailloux. "The Z-Wave routing protocol and its security implications". In: *Computers & Security* 68 (2017), pp. 112–129. ISSN: 0167-4048. DOI: <https://doi.org/10.1016/j.cose.2017.04.004>.
 - [371] J. Haxhibeqiri, E. De Poorter, I. Moerman, and J. Hoebeke. "A Survey of LoRaWAN for IoT: From Technology to Application". In: *Sensors* 18.11 (2018). ISSN: 1424-8220. DOI: 10.3390/s18113995.
 - [372] R. Sedgewick and K. Wayne. *Algorithms*. Addison-Wesley Professional, 2011.
 - [373] L. Burkholder. "The Halting Problem". In: *ACM SIGACT News* 18.3 (1987), pp. 48–60.
 - [374] *Hue White and Color Ambiance A60 - E27 smart bulb - 800*. Accessed: December 5, 2024. Philips. 2024. URL: <https://www.philips-hue.com/en-my/p/hue-white-and-color-ambiance-a60---e27-smart-bulb---800/8718696725580>.
 - [375] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988. ISBN: 1558604790.
 - [376] R. J. Rossi. *Mathematical statistics: an introduction to likelihood based inference*. John Wiley & Sons, 2018.
 - [377] D. Sankoff and J. B. Kruskal. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. David Hume series. Center for the Study of Language and Information, 1999. ISBN: 1575862174.
 - [378] M. Franceschinis, C. Pastrone, M. A. Spirito, and C. Borean. "On the performance of ZigBee Pro and ZigBee IP in IEEE 802.15.4 networks". In: *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2013, pp. 83–88. DOI: 10.1109/WiMOB.2013.6673344.
 - [379] S. Sistu, Q. Liu, T. Ozcelebi, E. Dijk, and T. Zotti. "Performance Evaluation of Thread Protocol based Wireless Mesh Networks for Lighting Systems". In: *2019 International Symposium on Networks, Computers and Communications (ISNCC)*. 2019, pp. 1–8. DOI: 10.1109/ISNCC.2019.8909109.
 - [380] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal. "Chaos Engineering". In: *IEEE Software* 33.3 (2016), pp. 35–41. DOI: 10.1109/MS.2016.60.

- [381] *chaosmonkey*. Accessed: July 24, 2025. Netflix. Oct. 2024. URL: <https://github.com/Netflix/chaosmonkey>.
- [382] *Chaos Mesh*. Accessed: July 24, 2025. Cloud Native Computing Foundation. July 2025. URL: <https://chaos-mesh.org>.
- [383] *Steadybit/Chaos Engineering Tool for Reliability Testing*. Accessed: July 24, 2025. Steadybit. July 2025. URL: <https://steadybit.com>.
- [384] *How Google Play works*. Accessed: February 5, 2026. Google. URL: <https://google.play/howplayworks>.
- [385] *App Store*. Accessed: February 5, 2026. Apple Inc. 2026. URL: <https://www.apple.com/app-store>.
- [386] J. F. Lemmer and D. E. Gossink. "Recursive noisy OR - a rule for estimating complex probabilistic interactions". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34.6 (2004), pp. 2252–2261. DOI: 10.1109/TSMCB.2004.834424.
- [387] *Omada Cloud's domain names*. Accessed: January 20, 2026. TP-Link. 2024. URL: <https://www.tp-link.com/en/support/faq/3542>.
- [388] B. Wolford. *What is GDPR, the EU's new data protection law?* Accessed: October 16, 2025. 2025. URL: <https://gdpr.eu/what-is-gdpr>.
- [389] B. Gruver. *smali*. Accessed: October 4, 2025. 2025. URL: <https://github.com/JesusFreke/smali>.
- [390] *Apktool*. Accessed: October 4, 2025. 2025. URL: <https://apktool.org>.
- [391] *skylot. JADX*. Accessed: October 4, 2025. 2025. URL: <https://github.com/skylot/jadx>.
- [392] *Socket (Java Platform SE 8)*. Accessed: February 24, 2026. Oracle. 2025. URL: <https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html>.
- [393] X. Wang, Y. Sun, S. Nanda, and X. Wang. "Looking from the Mirror: Evaluating IoT Device Security through Mobile Companion Apps". In: 2019, pp. 1151–1167. ISBN: 978-1-939133-06-9. (Visited on 02/24/2026).
- [394] J. Kim, H. Choi, H. Namkung, W. Choi, B. Choi, H. Hong, Y. Kim, J. Lee, and D. Han. "Enabling Automatic Protocol Behavior Analysis for Android Applications". In: *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '16. Irvine, California, USA: Association for Computing Machinery, 2016, pp. 281–295. ISBN: 9781450342926. DOI: 10.1145/2999572.2999596.

- [395] S. Rasthofer, S. Arzt, and E. Bodden. “A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks”. In: *NDSS*. Vol. 14. 1125. 2014.
- [396] G. Piskachev, L. N. Q. Do, and E. Bodden. “Codebase-adaptive detection of security-relevant methods”. In: *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2019. Beijing, China: Association for Computing Machinery, 2019, pp. 181–191. ISBN: 9781450362245. DOI: 10.1145/3293882.3330556.
- [397] Androguard. *Welcome to the Androguard Docs!* Accessed: October 8, 2025. androguard. 2024. URL: <https://androguard.github.io/androguard>.
- [398] J. Hoffmann, M. Ussath, T. Holz, and M. Spreitzenbarth. “Slicing droids: program slicing for smali code”. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. SAC ’13. Coimbra, Portugal: Association for Computing Machinery, 2013, pp. 1844–1851. ISBN: 9781450316569. DOI: 10.1145/2480362.2480706.
- [399] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. “FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps”. In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI ’14. Edinburgh, United Kingdom: Association for Computing Machinery, 2014, pp. 259–269. ISBN: 9781450327848. DOI: 10.1145/2594291.2594299.
- [400] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. “Energy-efficient communication protocol for wireless microsensor networks”. In: *Proceedings of the 33rd annual Hawaii international conference on system sciences*. IEEE. 2000, 10–pp.
- [401] *NetSim Standard: Accelerate Network R&D*. Accessed: October 23, 2025. Tetcos. URL: <https://www.tetcos.com/netsim-std.html>.
- [402] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. “Cross-Level Sensor Network Simulation with COOJA”. In: *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*. 2006, pp. 641–648. DOI: 10.1109/LCN.2006.322172.

Appendix

A

A.1 Summary of all surveyed datasets

In the following, we summarize all datasets we covered for the work described in Chapter 3. For each, we identify various characteristics, roughly following the FODA taxonomy defined in Section 3.4. Those characteristics are defined in the following. For two of them, we also described the abbreviations used throughout the tables.

- *Year*: Year(s) in which the dataset was recorded
- *Protocol*: Communication protocol(s) used by the devices leveraged to produce the dataset
- *Data*: Format of the data contained in the dataset
- *Devices*: Number and category of devices leveraged to produce the dataset
 - GP: General purpose (PCs, smartphones, servers, etc.)
 - SH: Smart Home
 - sen.: Sensors (& actuators)
 - GW: Gateway
 - S: Servers
 - HP: Honeytrap
 - BN: Botnet
 - wear.: Wearables (e.g. smart watches)
- *Duration*: Total recording duration
- *Methodology*: Strategy implemented to produce the dataset
 - R: Real-world
 - T: Testbed
 - E: Emulation

- S: Simulation
- H: Hybrid
- *Attacks?*: Whether the dataset comprises data related to malicious events
- *Features*: Whether the dataset contains descriptive data features, and, if relevant, their number
- *Labelled*: Whether the dataset is labelled
- *Availability*: Whether the dataset is publicly available
- *Used?*: Whether the dataset has been used by other researchers

Two additional, generic abbreviations are used in different columns of the tables:

- */*: Not relevant
- **U**: Unspecified

Tables A.1 and A.2 show the IoT-related datasets, *i.e.* the ones created by leveraging an environment comprising IoT devices. The former covers the ones which communicate using typical IP-based protocols, whereas the latter involves the ones which use IoT-specific protocols, such as the ones described in Section 2.1.3. Table A.3 covers the datasets not directly related to IoT networks, but which have been used in the context of IoT research, or might be relevant for such.

Dataset	Year	Protocol	Data	Devices
AWID2 [246]	2015	Wi-Fi	Features	7 GP
Sivanathan [208]	2016	IP	Full packets	4+ SH 3 GP
USC [209]	2016	U	Sensor data	4060 sen.
UNSW benign [250]	2016- 2017	IP	Full packets	28 SH
IoT SENTINEL [249]	2017	IP	Full packets	31 SH
DS2OS [213]	2018	U	Sensor data	U
N-BaIoT [211]	2018	IP	Full packets	9 SH 3 GP
Doshi [212]	2018	IP	Full packets	3 SH 1 GP
UNSW attacks [214]	2018	IP	Full packets	10 SH 4 GP
UNSW MUD [251]	2018	MUD	Device signatures	28 SH
Bezerra [210]	2018	IP	Full packets & Hardware	5 GP
Ren [256]	2018- 2019	IP	Full packets	81 SH 4 GP
IoTLS [99]	2018- 2020	IP TLS	Full packets	40 SH
Kang [186]	2019	IP	Full packets	2 SH
Saidi [252]	2019	IP	Device signatures	96 SH
Bot-IoT [215]	2019	IP MQTT	Full packets	U
IoT-23 [216]	2020	IP	Full packets	3 SH 1 GP
MedBIoT [218]	2020	IP	Full packets	83 SH 6 GP
Sarica [219]	2020	IP	Full packets	10 SH 8 GP
UNSW IPFIX [253]	2020	IPFIX	Flow stats	29 SH

Table A.1: IoT-related datasets involving IP-based protocols
Columns *Year* $\rightarrow$ *Devices*

Dataset	Duration	Methodology	Attacks?	Features	Labelled	Availability	Used?
AWID2 [246]	1 h	T	✓	✓(155)	✓	✓	✓
Sivanathan [208]	18 min	T	✓	✗	✗	✗	?
USC [209]	1 mo	R	✓	✗	✓	✓	?
UNSW benign [250]	6 mo 13 d	T	✗	✗	✗	✓	✓
IoT SENTINEL [249]	U	T	✗	✗	✗	✓	✓
DS2OS [213]	24 h	S	✓	✓(11)	✓	✓	✓
N-BaIoT [211]	U	T	✓	✓(23)	✓	✓	✓
Doshi [212]	Benign: 10 min Malicious: U	T	✓	✓(11)	✓	✗	?
UNSW attacks[214]	16 d	T	✓	✓(20)	✓	✓	✓
UNSW MUD [251]	6 mo	T	/	/	/	✓	✓
Bezerra [210]	9+ h	E	✓	✓(6)	✓	✓	✓
Ren [256]	4 d 16 h	T	✗	✗	✓	✓	✓
IoTLS [99]	~ 2 y	T	✗	✗	✗	✓	?
Kang [186]	U	T	✓	✗	✓	✓	✓
Saidi [252]	7 d	T	✓	/	/	✓	?
Bot-IoT [215]	U	E	✓	✓(32)	✓	✓	✓
IoT-23 [216]	21.5 d	T	✓	✗	✓	✓	✓
MedBIoT [218]	U	H	✓	✓(100)	✓	✓	✓
Sarica [219]	~ 1.5 d	S	✓	✓(33)	✓	✓	?
UNSW IPFIX [253]	3 mo	T	✗	✓(28)	✗	✓	?

Table A.1: IoT-related datasets involving IP-based protocols (continued)
Columns *Duration* → *Used?*

Dataset	Year	Protocol	Data	Devices
DAD [217]	2020	IP MQTT	Full packets	16 sen.
TON_IoT [220]	2021	IP	Full packets	U
27-Botnet [224]	2021	IP	Full packets	10 SH (27 BN)
Kolcun [254]	2021	IP	Full packets	41 SH 1 S
ECU-IoHT [222]	2020	IP	Header only	3 sen. 2 GP 1 GW
MQTTset [223]	2020	IP MQTT	Full packets	8 sen. 1 GP 1 GW
CCD-INID-V1 [221]	2021	IP	Full packets	4 sen.
ECU-IoFT [225]	2022	Wi-Fi	Features	1 drone 2 GP

Table A.1: IoT-related datasets involving IP-based protocols (continued)
Columns *Year* $\rightarrow$ *Devices*

Dataset	Duration	Methodology	Attacks?	Features	Labelled	Availability	Used?
DAD [217]	7 d	E	✓	✗	✓	✓	✓
TON_IoT [220]	U	E	✓	✓(44)	✓	✓	✓
27-Botnet [224]	18 d	T	✓	✓(23)	✓	✗	?
Kolcun [254]	27 w	T	✗	✓(37)	✓	✓	?
ECU-IoHT [222]	~ 3 mo 20 d	T	✓	✓(18)	✓	✓	?
MQTTset [223]	1 w	E	✓	✓(34)	✓	✓	✓
CCD-INID-V1 [221]	U	E	✓	✓(83)	✓	✗	?
ECU-IoFT [225]	3 min	T	✓	✓(10)	✓	✓	?

Table A.1: IoT-related datasets involving IP-based protocols (continued)
Columns *Duration* $\rightarrow$ *Used?*

Dataset	Year	Protocol	Data	Devices
LWSNDR [226]	2010	802.15.4	Sensor data	4 sen. 2 routers
Fu [247]	2015	802.15.4	Perf. metrics	2 sen.
BLEBeacon [188]	2016	BLE	Signal data	46 sen. 32 scanners
WSN-DS [227]	2016	LEACH [400]	Full packets	100 sen.
Dowling [187]	2017	Zigbee	Attack logs	1 HP
LP-WAN loc. [189]	2017- 2018	LoRa, Sigfox	Signal data Location	20+ sen.
Purnama [255]	2018	IP, Zigbee	Full packets	5 sen.
RPL-NIDDS- 17 [228]	2019	6LoWPAN RPL	Full packets	U
GHOST-IoT [194]	2019	IP, BT, Zigbee, RF869	Header only	9 SH 3 GP 2 GW
IoT-DDoS [229]	2019	802.15.4, RPL, 6LoWPAN, UDP	Full packets	29 sen.
LoED [190]	2019- 2020	LoRa	Full packets Signal data	8503 sen. 9 GW
cmu/zigbee- smarthome [193]	2020	Zigbee	Full packets	10 SH 1 GW
Sadikin [230]	2020	Zigbee	Full packets	3 SH 1 GW
Chen [231]	2020	IP, BT, ZB, LoRa	Header only	15 sen.
LoRa radio (2020) [191]	2020	LoRa	Signal data	100 sen.
Barman [257]	2020	BT/BLE	Full packets	13 wear.
cmu/zigbee- eda [195]	2021	Zigbee	Full packets	4 SH 1 GW
IoT-CIDDS [233]	2021	UDP, RPL, 6LoWPAN	Features	10-100 sen.
Lacava [232]	2021	BLE Mesh	Full packets	13 sen.
LoRa radio (2021) [192]	2021	LoRa	Signal data	25 sen.

Table A.2: IoT-related datasets involving IoT-specific protocols
Columns *Year* $\rightarrow$ *Devices*

Dataset	Duration	Methodology	Attacks?	Features	Labelled	Availability	Used?
LWSNDR [226]	6 h	T	✗	✗	✓	✓	✓
Fu [247]	6 mo	T	✗	✗	✗	✓	✓
BLEBeacon [188]	33 d	T	✗	✗	✗	✓	✓
WSN-DS [227]	1 h	S (ns-2 [178])	✓	✓(23)	✓	✓	✓
Dowling [187]	3 mo	T	✓	✗	✗	✗	?
LP-WAN loc. [189]	2 mo 21 d	T	✗	✗	✗	✓	✓
Purnama [255]	50 min	T	✓	✗	✗	✗	?
RPL-NIDDS -17 [228]	U	S (NetSim [401])	✓	✓(20)	✓	✗	✓
GHOST-IoT [194]	10 d	T	✓	✓(112)	✗	✓	✓
IoT-DDoS [229]	1 d	S (Cooja [402])	✓	✓(16)	✓	✗	?
LoED [190]	2-4 mo	R	✗	✗	✗	✓	✓
cmu/zigbee- smarthome [193]	36 h	T	✓	✗	✗	✓	✓
Sadikin [230]	U	T	✓	✗	✓	✗	?
Chen [231]	73 min	T	✓	✗	✓	✓	?
LoRa radio (2020) [191]	1+ d	T	✗	✗	✓	✓	?
Barman [257]	98 h	T	✓	✗	✗	/	✓
cmu/zigbee- eda [195]	4 d	T	✓	✗	✗	✓	?
IoT-CIDDS [233]	24 h	S (Cooja [402])	✓	✓(21)	✓	✓	?
Lacava [232]	6 h	T	✓	✗	✗	✓	?
LoRa radio (2021) [192]	5 d	T	✗	✗	✓	✓	?

Table A.2: IoT-related datasets involving IoT-specific protocols (continued)
Columns *Duration* → *Used?*

Dataset	Year	Protocol	Data	Devices
WUSTL-IIOT-2021 [196]	2021	Modbus	Features	7 sen. 4 GP 1 PLC
X-IIoTID [234]	2021	Modbus, CoAP, MQTT, etc.	Features	6 sen. 13 GP 1 PLC
Edge-IIoTset [235]	2022	Modbus, MQTT, etc.	Full packets	13 sen.
CoAP-DoS [197]	2022	CoAP	Full packets	4 GP

Table A.2: IoT-related datasets involving IoT-specific protocols (continued)
Columns *Year* → *Devices*

Dataset	Duration	Methodology	Attacks?	Features	Labelled	Availability	Used?
WUSTL-IIOT-2021 [196]	53 h	T	✓	✓(41)	✓	✓	✓
X-IIoTID [234]	114 d	H	✓	✓(67)	✓	✓	✓
Edge-IIoTset [235]	50 d	H	✓	✓(61)	✓	✓	✓
CoAP-DoS [197]	16 h	E	✓	✓(17)	✓	✓	?

Table A.2: IoT-related datasets involving IoT-specific protocols (continued)
Columns *Duration* → *Used?*

Dataset	Year	Protocol	Data	Devices
Dartmouth [198]	2009	IP	Header only	Thousands
CAIDA [199]	2007	IP	Header only	U
NSL-KDD [171]	2009	IP	Full packets	U
CDX 2009 [200]	2009	IP	Full packets	U
DARPA 2009 [201]	2009	IP	Full packets	U
Waikato [202]	Various	IP	Header only	U
Kyoto 2006+ [203]	2006-2015	IP	Header only	348 HP
SSENet-2011 [236]	2011	IP	Full packets	1 S
ADFA [237]	2012	/	System calls	1 S
ISCXIDS2012 [238]	2012	IP	Flow stats	21 PCs
TUIDS [204]	2012	IP	Full packets	40 PCs
CTU-13 [205]	2013	IP	Full packets	U
UNSW-NB15 [206]	2015	IP	Full packets	3 S
UGR'16 [239]	2016	IP	Flow stats	25 PCs
CIC-IDS-2017 [240]	2017	IP	Full packets	14 PCs 3 S
CSE-CIC-IDS-2018 [241]	2018	IP	Full packets	420 PCs 30 S
CICDDoS-2019 [242]	2019	IP	Full packets	4 PCs 1 S
CIRA-CIC-DoH-Brw-2020 [243]	2020	HTTPS	Flow stats	14 S
LITNET [244]	2020	IP	Flow stats	5 GW
AWID3 [248]	2021	Wi-Fi	Full packets	11 GP 6 S
NF-UQ-NIDS [245]	2021	IP	Flow stats	/
H23Q [207]	2022	HTTP QUIC	Full packets	13 PCs 6 S

Table A.3: Non-IoT datasets
Columns Year → Devices

Dataset	<i>Duration</i>	<i>Methodology</i>	<i>Attacks?</i>	<i>Features</i>	<i>Labelled</i>	<i>Availability</i>	<i>Used?</i>
Dartmouth [198]	5+ y	R	✗	✗	✗	✓	✓
CAIDA [199]	1 h	R	✓	✗	✗	✓	?
NSL-KDD [171]	U	S	✓	✗	✓	✓	✓
CDX 2009 [200]	4 d	E	✓	✗	✓	✓	✓
DARPA 2009 [201]	10 d	S	✓	✗	✗	✓	✓
Waikato [202]	Various	R	✗	✗	✗	✓	✓
Kyoto 2006+ [203]	9 y	E	✓	✓(24)	✓	✓	✓
SSENet-2011 [236]	4 h	T	✓	✓(19)	✓	✗	✓
ADFA [237]	U	E	✓	✗	✗	✓	✓
ISCXIDS2012 [238]	7 d	T	✓	✗	✓	✓	✓
TUIDS [204]	4 w	T	✓	✓(74)	✓	✗	✓
CTU-13 [205]	100+ h	E	✓	✓(22)	✓	✓	✓
UNSW-NB15 [206]	31 h	E	✓	✓(61)	✓	✓	✓
UGR'16 [239]	21 w	R	✓	✗	✓	✓	✓
CIC-IDS-2017 [240]	5 d	T	✓	✓(80)	✓	✓	✓
CSE-CIC-IDS-2018 [241]	10 d	T	✓	✓(83)	✓	✓	✓
CICDDoS-2019 [242]	14 h 40 min	T	✓	✓(80+)	✓	✓	✓
CIRA-CIC-DoH-Brw-2020 [243]	U	T	✓	✓(28)	✓	✓	✓
LITNET [244]	10 mo	R	✓	✓(85)	✓	✓	✓
AWID3 [248]	2 h 20 min	T	✓	✓(254)	✓	✓	✓
NF-UQ-NIDS [245]	/	/	✓	✓(83)	✓	✓	✓
H23Q [207]	1 h 47 min	T	✓	✓(200)	✓	✓	✗

Table A.3: Non-IoT datasets (continued)
Columns *Duration* → *Used?*

A.2 Complete device profiles

In Chapter 5, we presented the IETF MUD standard, a format to express the expected communication patterns of IoT devices. We proposed a new format, more expressive, inspired by MUD. In this appendix, we present a sample full profile for both formats.

A.2.1 Manufacturer Usage Description (MUD)

```

1  {
2    "ietf-mud:mud": {
3      "mud-version": 1,
4      "mud-url": "http://192.0.2.1/mud/tplinkplugMud.json",
5      "mud-signature": "http://192.0.2.1/mud/tplinkplugMud.p7s",
6      "last-update": "2026-01-01T10:00:00.000+00:00",
7      "cache-validity": 100,
8      "is-supported": true,
9      "systeminfo": "tplink-plug",
10     "from-device-policy": {
11       "access-lists": {
12         "access-list": [
13           {
14             "name": "from-ipv4-tplinkplug"
15           }
16         ]
17       }
18     },
19     "to-device-policy": {
20       "access-lists": {
21         "access-list": [
22           {
23             "name": "to-ipv4-tplinkplug"
24           }
25         ]
26       }
27     }
28   },
29   "ietf-access-control-list:access-lists": {
30     "acl": [
31       {
32         "name": "from-ipv4-tplinkplug",
33         "type": "ipv4-acl-type",
34         "aces": {
35           "ace": [
36             {
37               "name": "from-plug-phone-udp",

```

```

38     "matches": {
39         "ipv4": {
40             "protocol": 17,
41             "destination-ipv4-network": "192.168.1.2/32"
42         },
43         "udp": {
44             "source-port": {
45                 "operator": "eq",
46                 "port": 9999
47             }
48         }
49     },
50     "actions": {
51         "forwarding": "accept"
52     }
53 },
54 {
55     "name": "from-plug-phone-tcp",
56     "matches": {
57         "ipv4": {
58             "protocol": 6,
59             "destination-ipv4-network": "192.168.1.2/32"
60         },
61         "tcp": {
62             "source-port": {
63                 "operator": "eq",
64                 "port": 9999
65             }
66         }
67     },
68     "actions": {
69         "forwarding": "accept"
70     }
71 },
72 {
73     "name": "from-plug-tplinkcloud-https",
74     "matches": {
75         "ipv4": {
76             "protocol": 6,
77             "ietf-acldns:dst-dnsname": "n-devs.tplinkcloud.com"
78         },
79         "tcp": {
80             "destination-port": {
81                 "operator": "eq",
82                 "port": 443
83             }
84         }
85     }
86 }

```

```

85         },
86         "actions": {
87             "forwarding": "accept"
88         }
89     }
90 ]
91 }
92 },
93 {
94     "name": "to-ipv4-tplinkplug",
95     "type": "ipv4-acl-type",
96     "aces": {
97         "ace": [
98             {
99                 "name": "to-plug-phone-tcp",
100                 "matches": {
101                     "ipv4": {
102                         "protocol": 6,
103                         "source-ipv4-network": "192.168.1.2/32"
104                     },
105                     "tcp": {
106                         "destination-port": {
107                             "operator": "eq",
108                             "port": 9999
109                         }
110                     }
111                 },
112                 "actions": {
113                     "forwarding": "accept"
114                 }
115             },
116             {
117                 "name": "to-plug-tplinkcloud-https",
118                 "matches": {
119                     "ipv4": {
120                         "protocol": 6,
121                         "ietf-acldns:src-dnsname": "n-devs.tplinkcloud.com"
122                     },
123                     "tcp": {
124                         "source-port": {
125                             "operator": "eq",
126                             "port": 443
127                         }
128                     }
129                 },
130                 "actions": {
131                     "forwarding": "accept"

```

```

132     }
133   }
134 ]
135 }
136 }
137 ]
138 }
139 }

```

A.2.2 Our format

```

1  device-info:
2    name: tplink-plug
3    mac: aa:bb:cc:dd:ee:ff
4    ipv4: 192.168.1.2
5    network: wireless
6
7
8  single-policies:
9
10   # Background UDP traffic between phone in LAN and plug
11   udp-plug-phone:
12     protocols:
13       udp:
14         src-port: 9999
15       ipv4:
16         src: self
17         dst: phone
18
19   # TCP traffic between phone in LAN and plug
20   tcp-device-plug-lan:
21     protocols:
22       tcp:
23         dst-port: 9999
24       ipv4:
25         src: phone
26         dst: self
27     bidirectional: true
28     stats:
29       rate: 20/second
30
31
32  interactions:
33
34   # DNS + HTTPS towards cloud (domain n-devs.tplinkcloud.com)

```

```
35 dns-https-tplinkcloud:
36
37 dns-query-plugin-tplinkcloud:
38 protocols:
39 dns:
40 qtype: A
41 domain-name: n-devs.tplinkcloud.com
42 udp:
43 dst-port: 53
44 ipv4:
45 src: self
46 dst: gateway
47 bidirectional: true
48
49 https-plugin-tplinkcloud:
50 protocols:
51 tcp:
52 dst-port: 443
53 ipv4:
54 src: self
55 dst: n-devs.tplinkcloud.com
56 bidirectional: true
57 stats:
58 rate: 10/second
```

A.3 Firewall-incurred latency: split per device

Figure A.1 shows a violin plot of the latency incurred by the firewall described in Chapter 5, in attack-free scenarios, split per device.

We observe a similar distribution of latencies for all devices: the activation of our firewall incurs a higher latency, due to more important processing, which is even more substantial when the firewall accounts for all devices at the same time. Nevertheless, said latency stays, in the vast majority of cases, negligible compared to the network latency (2 to 5 ms in a LAN).

The higher latency observed for the Philips Hue device can be explained by the fact that this device transmits packets that require more processing, namely a lookup in the DNS cache.

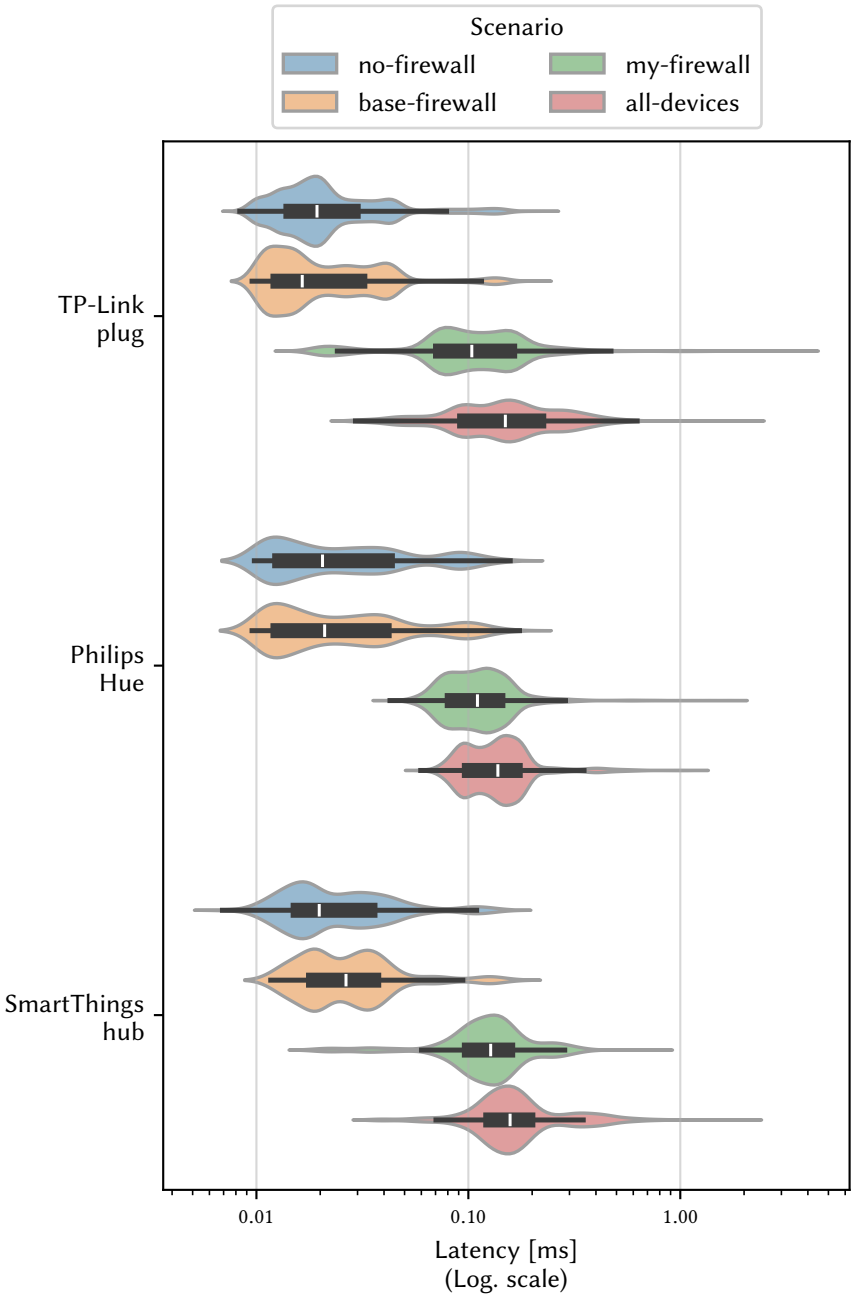


Figure A.1: Latency incurred by the AP hosting the firewall, in attack-free scenarios, split per device

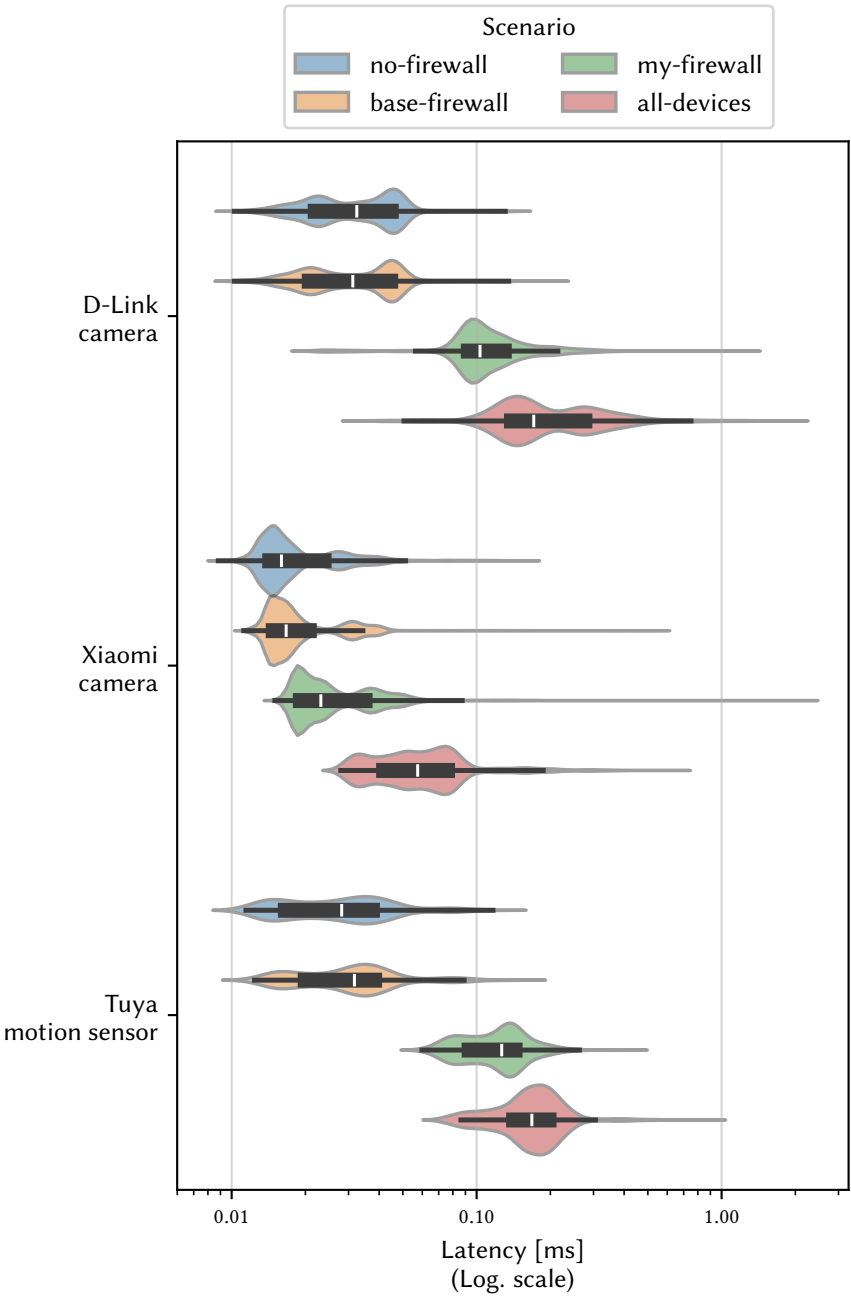


Figure A.1: Latency incurred by the AP hosting the firewall, in attack-free scenarios, split per device (continued)