POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Politecnico
di Torino

Master's Degree Thesis

Enhancing MUD policy management in a Smart Home Gateway environment

Supervisors

Prof. Fulvio CORNO

Dott. Luca MANNELLA

Candidate

Emanuele LINTY BLANCHET

December 2023

To my family, for always being at the center of their concerns.
To my friends, for accompanying me in this beautiful journey.
To myself, for never giving up.

Alla mia famiglia, per essere sempre stato al centro delle loro preoccupazioni.
Ai miei amici, per avermi accompagnato in questa bellissima esperienza.
A me, per non aver mai mollato.

Summary

The widespread use of Internet of Things (IoT) technology has brought unparalleled ease and comfort, but it has also triggered significant worries about security and privacy. The intricate and diversified nature of IoT ecosystems, encompassing various device categories and manufacturers, necessitates the formation of proper security measures to fend off potential threats.

One of the promising solutions to tackle this issue is the Manufacturer Usage Description (MUD) standard. MUD enables manufacturers to specify the endpoints that a MUD-enabled device can access, thus blocking all other unwanted connections. Its implementation enhances security and privacy in IoT environments, ensuring that devices only communicate with authorized endpoints.

This thesis aims to optimize a recently proposed approach to merge developer-specified endpoints into a single gateway-level MUD file for devices and for software plug-ins that lack inherent MUD support. With the ongoing expansion of IoT ecosystems comes the possibility of overlapping rules and conflicts that need addressing when merging several files. Our study tackles this challenge resolving conflicts and optimizing the rules within the gateway-level MUD file.

Nonetheless, it is crucial that the rules are written accurately to enable the MUD manager, which enforces the MUD policies, to process them without errors. This requires a meticulous inspection to confirm that every policy adheres to a valid structure while containing no forbidden elements according to MUD standards. Ensuring the integrity of these policies is crucial for maintaining the security and reliability of the IoT network.

The aim of this research is to produce a final MUD file that is conflict-free and optimized to enhance rule application speed and efficiency. This thesis demonstrates that, through optimization, the final number of rules is reduced, and with conflict management, there are no rules that can be interpreted ambiguously, resulting in a robust and secure framework for IoT device communication.

Acknowledgements

And so, in the blink of an eye, five years have passed. It feels like yesterday when I walked through the doors of Politecnico: the enthusiasm to learn was great, accompanied by a slight fear of failure. Today, I can proudly say that I have made it.

If I have reached this milestone, it is also thanks to those who have always supported me. Dad, Grandma, Monica: you have been my everyday life, always putting me first, and I will always be grateful to you.

A thank you also goes to all my friends, who have been an integral part of my journey. You have always been there, and I will never forget all the beautiful moments we spent together between Aosta and Turin.

I would also thank Professor Fulvio Corno and Doctor Luca Mannella for guiding me in these final months and for always being available for me.

And finally, my thoughts go to the most beautiful star in the sky: Mom, you would have been, and I am sure you are, proud of me.

Acknowledgements

E così, in un batter d'occhio, sono trascorsi cinque anni. Mi sembra ieri quando ho varcato le porte del Politecnico: la voglia di imparare era tanta, assieme però a una leggera paura di non riuscirci. Invece oggi posso dire di avercela fatta.

Se ho raggiunto questo traguardo, è anche grazie a chi mi ha sempre sostenuto. Papà, Nonna, Monica: siete stati la mia quotidianità, mi avete sempre messo in primo piano e ve ne sarò sempre grato.

Un ringraziamento va anche a tutti i miei amici, parte integrante del mio percorso: siete sempre stati presenti e mai dimenticherò tutti i bei momenti trascorsi con voi tra Aosta e Torino.

Ringrazio poi il professor Fulvio Corno e il Dottor Luca Mannella, per avermi guidato in questi mesi finali del mio percorso e per essere sempre stati disponibili con me.

E infine i miei pensieri vanno alla stella più bella del firmamento: Mamma, saresti stata, e sono sicuro che sei, fiera di me.

Table of Contents

List of Tables IX

List of Figures X

List of Listings XII

1 Introduction 1

2 Background 4
2.1 Policy-Based Management and Firewalls 4
2.2 Home Assistant 9
2.2.1 Terminology 9
2.2.2 Installation methods 10

3 Introducing Manufacturer Usage Description (MUD) in Smart Home Gateways 13
3.1 Manufacturer Usage Description 13
3.1.1 Terminology 14
3.1.2 MUD File Structure 14
3.1.3 Functional Overview 17
3.2 Extended MUD Architecture 18
3.2.1 Workflow and main components 18
3.2.2 MUD Aggregator 20
3.2.3 Limitations 21

4 Proposed Approach 23
4.1 Validation 23
4.2 Rule Relations 28
4.2.1 Optimizations 30
4.2.2 Conflicts 35

5 Experimental Results 43
5.1 Experimental setup 43
5.2 Validation 44
5.3 Optimizations 51
5.4 Conflicts 55
5.5 Discussion 59

6 Conclusions 61

Bibliography 65

A Setup of MUD Aggregator 70

List of Tables

2.1 Comparison between Home Assistant installation methods 12

5.1 Comparison results between `mud.yang.go` and `mud_json_validator.py`: IoTOPIA files 44
5.2 Comparison results between `mud.yang.go` and `mud_json_validator.py`: MUDGee files 47
5.3 Comparison time between `mud.yang.go` and `mud_json_validator.py`: IoTOPIA files 49
5.4 Comparison time between `mud.yang.go` and `mud_json_validator.py`: MUDGee files 51
5.5 Optimization results 52
5.6 Conflict resolution results 56

List of Figures

1.1 Estimated annual revenue generated by IoT 1
1.2 Smart home network 2

2.1 Flow of an ECA rule 4
2.2 “Once” trigger type 5
2.3 “Continuous” trigger type 5
2.4 “Switched” trigger type 6
2.5 Policy enforcement 6

3.1 MUD functional overview 18
3.2 Example of a possible Smart Home 19
3.3 Extended MUD-architecture 20

4.1 Flowchart of the validation process 24
4.2 Rules “exactly matching” 28
4.3 Rules “inclusively matching” 28
4.4 Rules “partially matching” 29
4.5 Rules “completely disjoint” 29
4.6 Rules “correlated” 29
4.7 Rules “inclusively matching” 30
4.8 The smallest rule is removed 31
4.9 Rules “exactly matching” 31
4.10 One rule is removed 32
4.11 Rules “partially matching” 33
4.12 A unique rule remains 33
4.13 Flowchart of rules anomalies resolution 36
4.14 Conflicted rules “partially matching” 37
4.15 Conflict resolved for rules “partially matching” (ATP and MSTP) . 39
4.16 Conflict resolved for rules “partially matching” (DTP and LSTP) . 40
4.17 Conflicted rules “inclusively matching” 42
4.18 Conflict resolved for rules “inclusively matching” (DTP and LSTP) 42

5.1 Two ACEs “exactly matching” 55
5.2 “Exactly matching” conflict resolution 55
5.3 Conflicted rules 59
5.4 Conflict resolved 59

List of Listings

3.1 Example of header fields of a MUD file 15
3.2 Example of `from-device-policy` 15
3.3 Example of `to-device-policy` 16
3.4 Example of an `Access Control Entry (ACE)` 16
4.1 One field for each possible value type 25
4.2 Example of some fields that must be present in the MUD file 25
4.3 Defining operator `enum` 26
4.4 Defining port field with a specified range 26
4.5 Defining destination-mac-address through regex 26
4.6 Example of control through regex 27
4.7 Example of two ACEs "partially matching" 34
4.8 Example of two ACEs "partially matching" optimized 35
4.9 Conflicted rules "partially matching" 37
4.10 Conflict resolved for rules "partially matching" (ATP and MSTP) . 39
4.11 Conflict resolved for rules "partially matching" (DTP and LSTP) . 40
5.1 `from-ipv4-withingsbabymonitor-7` 53
5.2 `from-ipv4-withingssleepsensor-5` 54
5.3 `from-ipv4-blipcarebpmeter-3` 57
5.4 `from-ipv4-ihomepowerplug-2` 57
A.1 Configuration code for integrating the *MUD Aggregator* in the `configuration.yaml` file 70
A.2 The `__init__` method 71
A.3 The `update` method 71

Chapter 1

Introduction

Year after year, the relevance of the Internet of Things (IoT) [1] continues to grow, with significant implications for everyday use and the economy. It is estimated that by 2030, the annual revenue generated by IoT will reach $621 billion, and this market is projected to continue expanding [2] (see Figure 1.1).

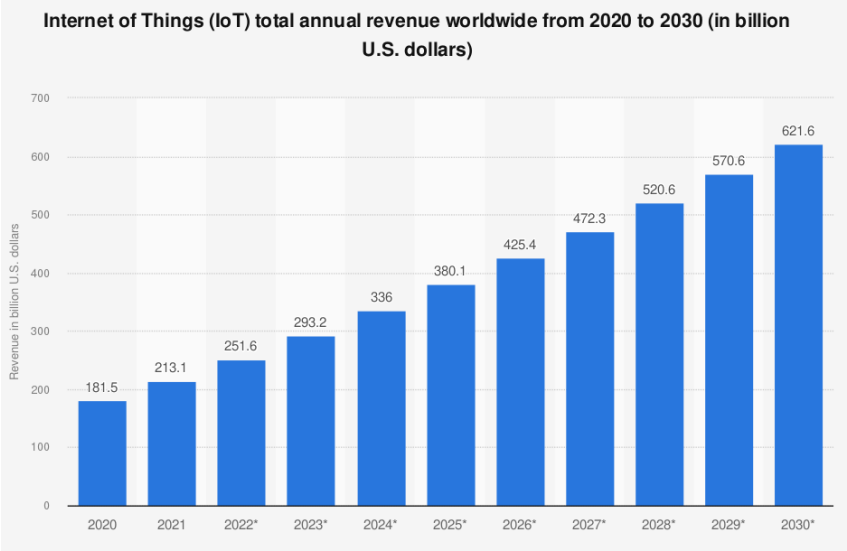

Figure 1.1: Estimated annual revenue generated by IoT

As the name suggests, IoT refers to a network of connected devices that exchange data and information over the internet. To be more specific, the IoT world can be divided into 5 categories [3]: Internet of Things (IoT), Internet of Everything (IoE), Internet of Nano Things (IoNT), Internet of Mission-Critical Things (IoMCT) and Internet of Mobile Things (IoMT).

In all areas, security plays a critical role: if a device is compromised, the entire

network in which it is located can be at risk, so it is necessary to pay attention to this aspect in order to avoid loss of privacy and theft of personal information.

One possible way to protect internet-connected devices is by using the Manufacturer Usage Description (MUD) standard [4]. This standard enables device developers to specify the network endpoints, both local and global, with which their products can interface and exchange data. The MUD approach utilizes whitelisting, wherein any communication that is not explicitly authorized is blocked. Each MUD-enabled device has its own MUD file in which the manufacturer specifies all network rules in JSON [5] format.

Given the assumption that we generally own and operate multiple electronic devices, it is advantageous to have a unified method of managing them and integrating their network policies.

To address the management challenge, a possible solution is Home Assistant [6]. Home Assistant is an open-source platform that enables users to manage all our Internet of Things (IoT) devices through integrations. Integrations are software components or plug-ins that act as a bridge from Home Assistant to the products.

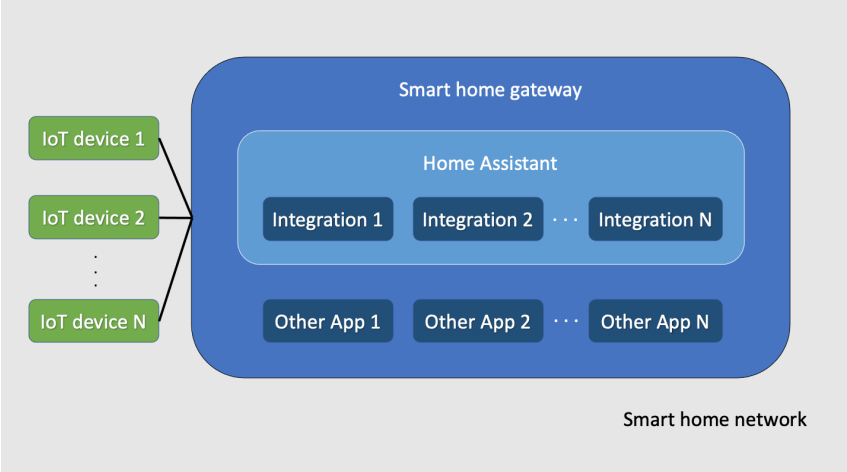

Figure 1.2: Smart home network

To address the policies integration, we have utilized *MUD Aggregator* [7], a custom integration designed for generating a gateway-level MUD file. This integration aggregates MUD snippets written by developers for integrations that do not support MUD into a gateway-level MUD file. MUD snippets are a subset of complete MUD files that follow the same syntax rules, but consist only of network policies, excluding other fields.

Merging MUD snippets into a single gateway-level MUD file can result in various issues that this thesis aims to resolve:

- First of all, a MUD snippet or the generated MUD file may contain syntax errors or values that are not permitted by the MUD standard.
- Then, since there are likely to be more MUD snippets, and thus more rules, overlapping policies may occur.

The overlapping policies can be divided in two categories:

- They can take the same action by either allowing or denying the specified network segment. Here, it is possible to perform optimization and to merge them in a unique rule.
- Some of them permit connections, while others refuse them. Here, it is necessary to resolve these conflicts, by choosing what is the action that the user wants to take.

The objective of this thesis is to analyze the integration *MUD Aggregator* and to modify it, by implementing the syntactical validation of MUD snippets and by improving the aggregation of network rules.

The structure of this work is as follows:

- Chapter 2, "Background": it highlights some preconceptions necessary to better understand the following chapters. For instance this chapter discusses the theory of policy management and the concept of smart home gateway, with a particular focus on Home Assistant.
- Chapter 3, "Introducing Manufacturer Usage Description (MUD) in Smart Home Gateways": here, the MUD Standard is analyzed in depth, together with how it's integrated with Home Assistant.
- Chapter 4, "Proposed Approach": in this chapter it is described how a MUD snippet is validated and how we've improved the aggregation of the plug-ins' policies, by optimizing and by resolving conflicts between those policies.
- Chapter 5, "Experimental Results": here, all the results of the experiments and the methods of achieving them are presented.
- Chapter 6, "Conclusions": it wraps up this thesis, resuming the work done in the previous chapters.

Chapter 2

Background

2.1 Policy-Based Management and Firewalls

Policy-Based Management (PBM) [8] is a comprehensive approach to managing and enforcing rules, regulations, and best practices within an organization or system. These policies serve as a set of of rules to administer, manage, and control access to network resources.

The basic block of this system is the Policy Rule, that must be in Event Condition Action (ECA) form [9]:

- on Event (a particular change in our system)
- if Condition (the necessary circumstances that define when the action should be executed)
- do Action (the description of what is to be done)

Figure 2.1: Flow of an ECA rule

When an event occurs, the condition is triggered and this process can evolve in three different ways:

- Once (see Figure 2.2): the condition is evaluated continuously, until it is true, then it is removed from the list of evaluations. The action is activated only once, when the condition in true.

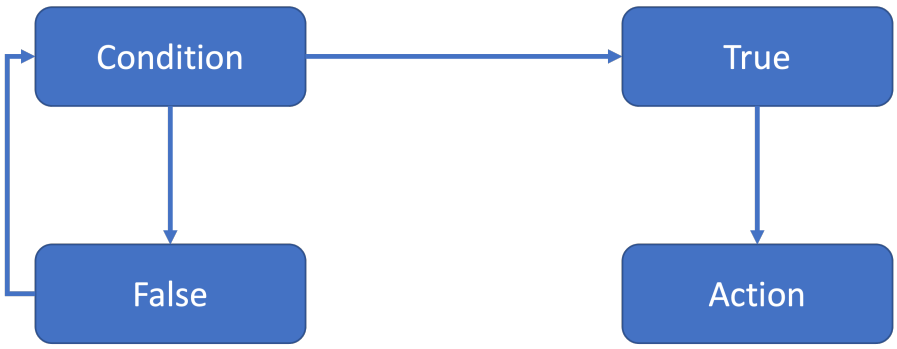

Figure 2.2: "Once" trigger type

- Continuous (see Figure 2.3): the condition is evaluated continuously and the action is repeated until the condition in true.

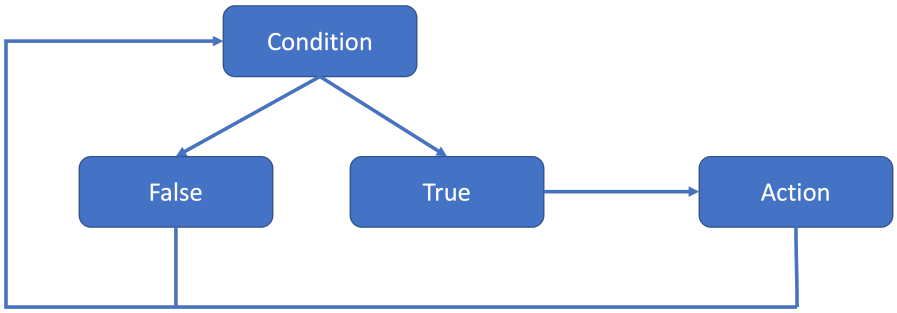

Figure 2.3: "Continuous" trigger type

- Switched (see Figure 2.4): the condition is evaluated continuously and the action is performed every time there is a switch between the condition true-false and false-true.

These decisions mentioned before are enforced by the Policy Enforcement Point (PEP) and they are taken by the Policy Decision Point (PDP) (see Figure 2.5), which can be:

- An approach that focuses on evaluating the conditions of a policy rule from a procedural perspective.
- An outcome-oriented approach that addresses the enforcement actions taken when the conditions of a policy rule are met.

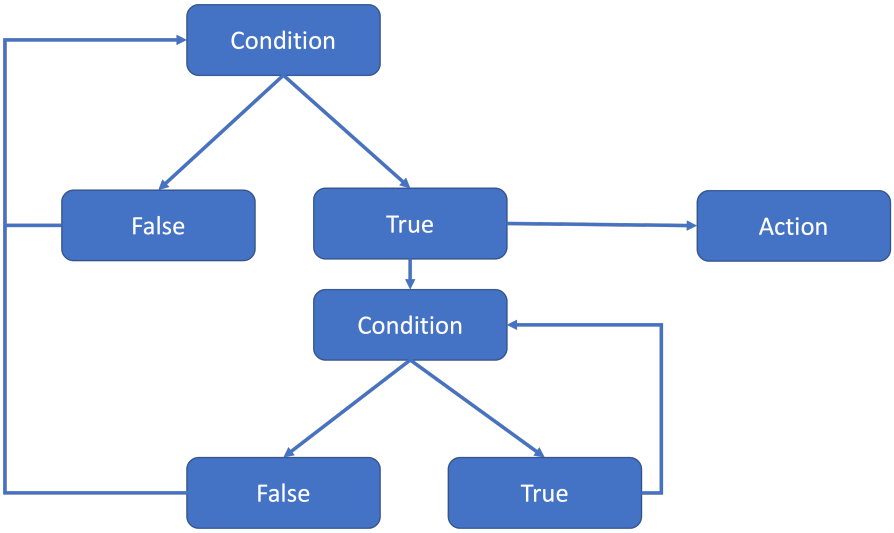

Figure 2.4: "Switched" trigger type

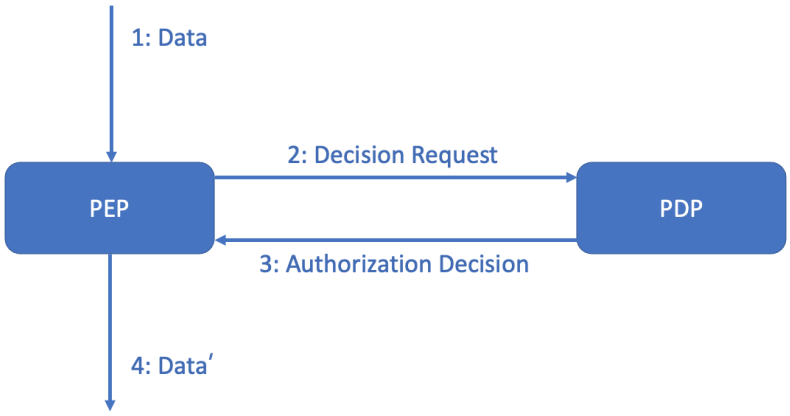

Figure 2.5: Policy enforcement

To better understand the following chapters, it is important to analyze the filtering policies, whose rules are composed of a condition (characterized by five network fields: protocol type, source and destination IP [10] addresses, source and destination ports) and an action (allow or deny).

The order of rules within a filtering policy is crucial. Rules are typically processed from the top to the bottom, with the first matching rule taking precedence. Therefore, the order of rules should be carefully arranged to ensure that the most

specific or important rules are evaluated first. According to [11], we can have five type of relationships between rules:

- They are "exactly matching" (EM): every field of R_x is the same of R_y. Formally, $R_x R_{EM} R_y$ iff

  $$\forall i : R_x[i] = R_y[i]$$

  where $i \in$ { protocol, src_ip, src_port, dst_ip, dst_port }

- They are "inclusively matching" (IM): they do not exactly match and every field in R_x is a subset or equal to the corresponding field in R_y. Formally, $R_x R_{IM} R_y$ iff

  $$\forall i : R_x[i] \subseteq R_y[i] \text{ and } \exists j \text{ such that: } R_x[j] \neq R_y[j]$$

  where $i, j \in$ { protocol, src_ip, src_port, dst_ip, dst_port }

- They are "partially matching" (PM): at least one field in R_x is a subset or a super-set or equal to the corresponding field in R_y and at least one field in R_x is not a subset and not a super-set and not equal to the corresponding field in R_y. Formally, $R_x R_{PM} R_y$ iff

  $$\exists i, j \text{ such that } R_x[i] \bowtie R_x[i] \text{ and } R_x[j] \not\bowtie R_x[j]$$

  where $\bowtie \in \{\subset, \supset, =\}$, $i, j \in$ { protocol, src_ip, src_port, dst_ip, dst_port }, $i \neq j$

- They are "completely disjoint" (CD): every field of R_x is different to the one in R_y. Formally, $R_x R_{CD} R_y$ iff

  $$\forall i : R_x[i] \not\bowtie R_y[i]$$

  where $\bowtie \in \{\subset, \supset, =\}$, $i \in$ { protocol, src_ip, src_port, dst_ip, dst_port }

- They are "correlated" (C): some fields in R_x are subsets or equal to the corresponding fields in R_y, and the rest of the fields in R_x are super-sets of the corresponding fields in R_y. Formally, $R_x R_C R_y$ iff

  $$\forall i : R_x[i] \bowtie R_y[i] \text{ and } \exists j, k \text{ such that: } R_x[j] \subset R_y[j] \text{ and } R_x[k] \supset R_y[k]$$

  where $\bowtie \in \{\subset, \supset, =\}$, $i, j, k \in$ { protocol, src_ip, src_port, dst_ip, dst_port }, $j \neq k$

To address these ambiguities and ensure that each packet matches only one rule, [12] suggests six different types of strategies:

- First Matching Rule (FMR): in an ordered list of rules, the action selected is the first.

- Last Matching Rule (LMR): in an ordered list of rules, the action selected is the second.
- Allow Takes Precedence (ATP): in case of different actions, the allow rule is preferred.
- Deny Takes Precedence (DTP): in case of different actions, the deny rule is preferred.
- Most Specific Takes Precedence (MSTP): the most specific rule is selected, the one that refers to the smaller number of ports.
- Least Specific Takes Precedence (LSTP): the least specific rule is selected, the one that refers to the larger number of ports.

Filtering policies are an important part of firewall management, defining the criteria that determine which network traffic is allowed or denied access to a network or system. Firewalls act as the gatekeepers, implementing these policies to filter incoming and outgoing data packets based on the condition mentioned before. Filtering policies can be tailored to meet specific security objectives, whether it's restricting access to sensitive data, protecting against malware, or complying with industry regulations. The key approaches that a firewall uses to protect a network are [13]:

- Packet Filtering: packet filtering firewalls work at the IP level, examining data packets as they pass through the firewall. This filtering method helps prevent unauthorized access and mitigates certain attacks but may not offer a robust defense against more sophisticated threats.
- Circuit Proxy: the second approach is to use a so-called circuit proxy. The main difference between this approach and the packet filtering firewall is that the former is the destination to which all communicators must address their packets. Once access has been allowed, the circuit proxy replaces the original address (its own) with the address of the intended recipient. The downside is that it requires the processing resources needed to modify the header.
- Application Proxy: this approach involves the use of an application proxy that understands the application protocol and data and is able to intercept information. It also has the ability to authenticate users and determine what type of data may pose a threat. The disadvantage is that users and client need to be often reconfigured.
- Packet Inspection: it is a more advanced form of firewall inspection. It not only examines packet headers but also analyzes the actual content of data packets. This granular analysis allows firewalls to detect and prevent a wide range of threats, such as malware and intrusion attempts.

2.2 Home Assistant

Home Assistant [6] is an open-source home automation platform that allows to control and automate various smart devices and services in our home. It serves as a central hub for managing and integrating a wide range of Internet of Things (IoT) devices, smart appliances, sensors, and software applications. The key features of Home Assistant are:

- Device Compatibility: Home Assistant is designed to work with a wide variety of smart devices, regardless of their brand or protocol. It supports popular technologies like Zigbee [14], Z-Wave [15], Wi-Fi [16], Bluetooth [17], and more.

- User-Friendly Interface: Home Assistant provides a web-based user interface that makes it easy to set up and manage devices and automation rules.

- Customization: it allows for extensive customization, including the creation of custom dashboards and user interfaces to suit our preferences and needs.

- Privacy and Local Control: Home Assistant emphasizes privacy and local control. It aims to keep data within our own network and not rely on cloud services for essential functions.

- Community and Development: Home Assistant has an active community of developers and users. This community continually contributes to the platform's growth and offers support through forums, documentation, and other resources.

2.2.1 Terminology

To better understand this chapter, here the most important parts of Home Assistant are listed[1]:

- Dashboards: they are customizable pages where the information regarding Home Assistant and regarding all our devices are displayed. There are two default Dashboards: Overview and Energy. The former is one that we see when we start Home Assistant and it gives the possibility to control the smart home, the latter, as the name suggests, displays the energy consumption.

[1] `https://www.home-assistant.io/getting-started/concepts-terminology/`, last visited on November 27th, 2023.

- Integrations: pieces of software with which Home Assistant can integrate with numerous third-party platforms and services, such as Amazon Alexa, Google Assistant, IFTTT, and various cloud services. Once an integration has been added, the hardware and/or the data are displayed in Home Assistant as devices and entities.
- Add-Ons: add-ons are typically third-party applications that can run with Home Assistant. Add-ons provide additional functionality, while integrations connect Home Assistant to other applications.
- Devices and Entities: a device is usually a physical product that can be controlled by Home Assistant through the corresponding integration. Entities are functional components or attributes of a device and allow us to interact with a specific feature of our product.
- Automations: they are a set of actions which can be set up to run automatically. They are characterized by three elements: triggers (event that starts the automations), conditions (that must occur so that the automations continue) and actions (the real interaction with the device).
- Scripts: they are very similar to the automations, but they don't have triggers, so they can't run unless they are used in automations.
- Scenes: they allow us to create predefined settings to apply to one ore more devices simultaneously, without having to manually interact with each product.

2.2.2 Installation methods

Home Assistant offers to the user four installation methods[2]:

- Home Assistant Core: it is the most flexible installation, providing the basic Home Assistant application without additional components or a supervisor. It is designed for advanced users who prefer full control over system configuration and do not require the additional services provided by the Supervisor. Users who choose Home Assistant Core typically have a good understanding of Python, virtual environments, and manual system configuration.
- Home Assistant Container: it allows users to run Home Assistant inside a Docker container. It is fundamental that the Home Assistant Core is executed inside a container. This type of installation provides a high level of flexibility because Docker containers encapsulate the application and its dependencies,

[2] `https://www.home-assistant.io/installation/`, last visited on November 10$^{\text{th}}$, 2023.

making it easy to deploy across different systems. Users who choose this method typically have experience with containerization technologies and prefer the ability to manage Home Assistant independently of the underlying operating system.

- Home Assistant Supervised: it is a middle-of-the-road option that combines the flexibility of a containerized installation with additional supervisor features for easier management. It runs Home Assistant inside a Docker container, but includes a supervisor that simplifies the installation of add-ons and manages system-related tasks. This type of installation is suitable for users who want a balance between flexibility and a more user-friendly experience without the constraints of a dedicated operating system.
- Home Assistant Operating System: it is a dedicated operating system designed to run Home Assistant as the primary and only application on the hardware. It provides a tight and secure environment that ensures Home Assistant has direct access to the underlying system resources. This type of installation is suitable for users who prefer a hands-off approach to system management, as it provides a complete, integrated solution with less manual configuration.

In addition to these characteristics, the comparison between the installation methods can be summarized as follows (see Table 2.1). From this table, it can be noticed that the integrations are available for all installation types, while the add-ons are only available for the Supervised installation method and for the HA Operating System. This is the main reason why this thesis focuses only on integrations.

	Core	Container	Supervised	HA OS
Automations	✓	✓	✓	✓
Dashboards	✓	✓	✓	✓
Integrations	✓	✓	✓	✓
Blueprints	✓	✓	✓	✓
Uses container	×	✓	✓	✓
Supervisor	×	×	✓	✓
Add-ons	×	×	✓	✓
Backups	✓	✓	✓	✓
Managed Restore	×	×	✓	✓
Managed OS	×	×	×	✓

Table 2.1: Comparison between Home Assistant installation methods

Chapter 3

Introducing Manufacturer Usage Description (MUD) in Smart Home Gateways

To improve the security of the Internet of Things (IoT), a promising standard is the Manufacturer Usage Description (MUD). In this chapter, we present the main features of this standard and its connection with the Smart Home Gateway that we used in our approach, i.e., Home Assistant.

3.1 Manufacturer Usage Description

The Manufacturer Usage Description (MUD) [4] is a standard that has the goal of improving the security of the Internet of Things (IoT), through the specification of the part of the network with which each device can communicate. Any other attempt at communication is blocked. In this chapter we will discuss that and we will present the structure of a MUD file.

The main objectives of MUD are as follows:

- Reduce the threat surface, allowing only traffic explicitly permitted by the manufacturer or by who writes the MUD file.
- Try to scale network policies to the increasing number of types of devices in the network.
- Address some vulnerabilities faster than updating the system itself. This means not replacing the update process, but adding an extra layer of protection.

- Define a standardized way for manufacturers to describe the intended communication behavior and requirements of their devices.

The MUD architecture consists of three blocks:

- A MUD URL useful for locating the product description.
- The MUD file description itself and how it can be interpreted.
- The network method to retrieve this description, i.e., the network protocol.

3.1.1 Terminology

In order to develop a more complete view of the standard MUD, in this section we present some key words related to this topic.

- MUD file: a JSON [5] file containing YANG-modeled [18] rules to describe the Thing network behavior.
- MUD file server: a web server where the MUD file is located.
- MUD manager: it has the task of retrieving the MUD file from the server and of making changes to network elements, after having processed it.
- MUD URL: a URL [19] used by the MUD manager to retrieve the MUD file.
- Thing: the device that emits the URL.
- Manufacturer: who configures the Thing to emit the MUD URL and who asserts a recommendation in a MUD file. The Manufacturer doesn't correspond always to who practically constructs the Thing, it can be a third party like a systems integrator or a component provider.

3.1.2 MUD File Structure

The MUD file is a key element in this context, because it is the place where the manufacturer, or who is in charge of doing that, specifies the network rules which characterize the device. It is written in YANG-based JSON and the key fields are listed in the following lines:

- `mud-version`: this field, as the name suggests, indicates the version of the MUD specification (see Listing 3.1).
- `last-update`: it specifies the date and the time of when the MUD file was generated or was modified for the last time (see Listing 3.1).

- `cache-validity`: this is the amount of time, in hours, that a network management station must wait since its last request before checking for an update (see Listing 3.1).
- `is-supported`: it is a boolean that indicates whether the Thing is supported or not. If it is not supported, it means that the manufacturer has no intention of ever releasing an update to the MUD file (see Listing 3.1).
- `mud-signature`: it is the URL that points to the signature of the MUD file (see Listing 3.1).
- `systeminfo`: it is a textual description of the device, useful for who doesn't know the product and wants a brief recap. It should not exceed 60 characters (see Listing 3.1).
- `documentation`: here we can instead have all the information about the device, it is an URL that points to the documentation related to the product and to the MUD file (see Listing 3.1).

Listing 3.1: Example of header fields of a MUD file

```
1    "mud-version":1,
2    "last-update":"2018-12-05T19:42:01+00:00",
3    "cache-validity":100,
4    "is-supported":true,
5    "mud-signature":
6        "https://example.com/us/p/F7C029.p7s",
7    "systeminfo":"This is an example device",
8    "documentation":
9        "https://example.com/doc/my-controller"
10
```

- `from-device-policy`: they describe the Access Lists (ACLs) present in the MUD file for outgoing connections (see Listing 3.2).

Listing 3.2: Example of `from-device-policy`

```
1    "from-device-policy":{
2       "access-lists":{
3          "access-list":[
4             {
5                "name":"mud-87176-v4fr"
6             }
7          ]
8       }
```

```
9     }
10
```

- `to-device-policy`: they describe the Access Lists (ACLs) present in the MUD file for incoming connections (see Listing 3.3).

Listing 3.3: Example of `to-device-policy`

```
1   "to-device-policy":{
2      "access-lists":{
3         "access-list":[
4            {
5               "name":"mud-87176-v4to"
6            }
7         ]
8      }
9   }
10
```

- `Access Control Entry (ACEs)`: they are the key element of a MUD file. In fact, they are the network rules to apply from the device to the rest of the world and vice versa (see Listing 3.4).

Listing 3.4: Example of an `Access Control Entry (ACE)`

```
1   "ace":[
2      {
3         "name":"to-ipv4-test-0",
4         "matches":{
5            "ietf-mud:mud":{
6               "local-networks":[
7                  null
8               ]
9            },
10           "ipv4":{
11              "protocol":17
12           },
13           "udp":{
14              "destination-port":{
15                 "operator":"eq",
16                 "port":39402
17              }
18           }
```

```
19            },
20            "actions":{
21                "forwarding":"accept"
22            }
23        }
24    ]
25
```

3.1.3 Functional Overview

In the previous sections we have done a brief introduction of the Manufacturer Usage Description (MUD) standard, highlighting some key terms and how a MUD file is constructed. Now it is fundamental to understand how the entire process works, how the rules are retrieved and how they are applied in our network.

Everything starts from the Thing, which exposes the MUD URL which points to the MUD file related to the device, stored in a MUD server. To have a secure communication and to retrieve correctly the file, the MUD URL must adhere to the "HTTPS" [20] standard. There are three ways a device can broadcast the MUD URL:

- DHCP [21] Option: The DHCP client sends a DHCP Request message including the MUD URL.

- X.509 [22] Constraint: The MUD URL is embedded in an X.509 certificate. It is an extension and it is non-critical. There are several ways to communicate this certificate, one of which is the Tunnel Extensible Authentication Protocol (TEAP) [23].

- Link Layer Discovery Protocol (LLDP) [24]: In this case, the MUD URL is included in a LLDP frame.

At this point, the router collects the MUD URL and forwards it to the MUD Manager, which can be part of the router or it can be an external entity. The MUD Manager is in charge of contacting the MUD File Server and of retrieving the MUD file. Finally, the device's policies are enforced: there isn't a default way to do this, it isn't specified in the standard. For example, the MUD Manager osMUD [25] uses *iptables* [26], a powerful and flexible program used for configuring and managing the packet filtering rules of the Linux kernel's [27] built-in firewall.

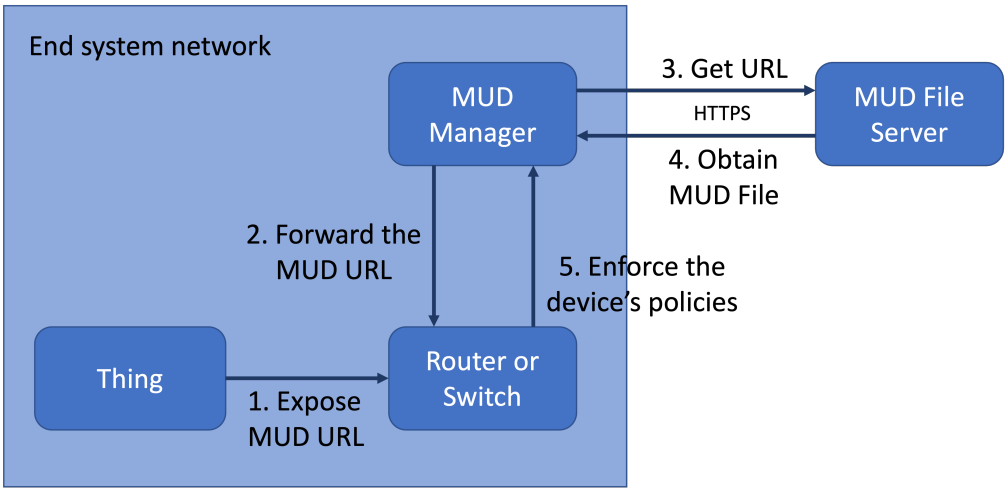

Figure 3.1: MUD functional overview

3.2 Extended MUD Architecture

To make MUD as effective as possible, it is important that all connected Things in a Smart Home follow the MUD standard. However, since currently not all devices are MUD-enabled by default, a recent study proposed a way to extend this capability for Smart Home Gateways. This section presents the characteristics of this work, along with the main components involved in generating, obtaining, and enforcing the rules described in a MUD file [28].

3.2.1 Workflow and main components

To extend the benefits of MUD to devices that are not MUD-enabled by default, the proposed solution [7] attaches a MUD snippet to the device plug-ins. A MUD snippet is written by plug-in developers and is a subset of the full MUD file: it follows the same syntax and it contains only the lists of Access Control Entries (ACEs), without the other keys associated with a full MUD file. In other words, it contains only the rules related to the plug-in.

For this solution to be successful, it requires an architecture capable of dynamically generating a MUD file from the MUD snippets of the plug-ins and enforcing the specified rules. The main components are:

- OpenWrt [29], an open-source, Linux-based operating system primarily designed for embedded devices, such as routers and access points.
- osMUD [25], an open-source MUD manager developed to be deployed on an

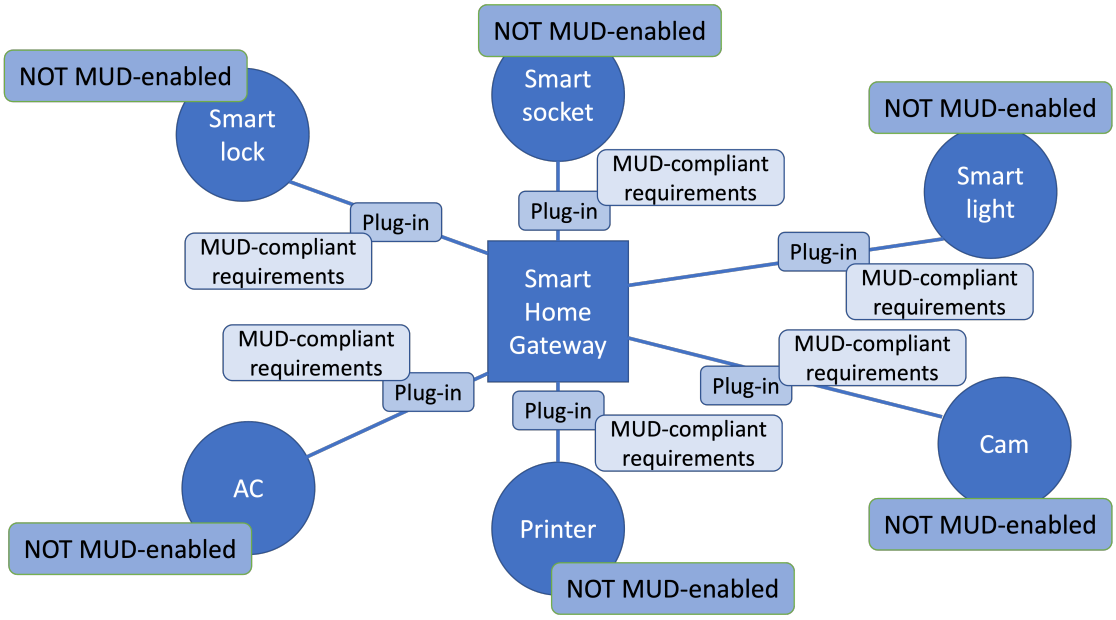

Figure 3.2: Example of a possible Smart Home

OpenWrt-based router.

- Home Assistant, an open-source Smart Home Gateway.
- *MUD Aggregator*[1], a Home Assistant integration built to create and expose a gateway-level MUD file.
- The gateway-level MUD file, a MUD file that contains all the rules included in the different MUD snippets of the plug-ins.
- The MUD policies, the requirements specified by developers in the MUD snippets.

Those components and their interactions are graphically described in figure 3.3. When Home Assistant starts, it runs all installed integrations and looks for MUD snippets associated with them. Each time a MUD snippet is found, its ACEs are collected and aggregated into the gateway-level MUD file. When all integrations are analyzed and the MUD file is complete, the MUD integration signs it with a private key to guarantee that the file came from the Smart Home Gateway. The aggregated MUD file is finally stored in a dedicated folder (step 1).

[1] `https://github.com/LucaMannella/HomeAssistant-MUD-Aggregator`, last visited on November 29th, 2023.

At this point the MUD file is ready to be exposed. This is done by the MUD integration by sending a *DHCP Request* message containing the MUD URL to the OpenWrt router where the osMUD MUD manager is running (currently osMUD only supports DHCP among all notification methods). This URL points to a web server able to expose the gateway-level MUD file (steps 2 and 3).

Finally, osMUD gets the MUD file (steps 4 and 5), verifies the signature and starts enforcing all the policies written in it: this is done through *iptables*, the default firewall of OpenWrt (step 6).

Every time that a new integration is added, the process should restart in order to keep the aggregated MUD file updated (step 7).

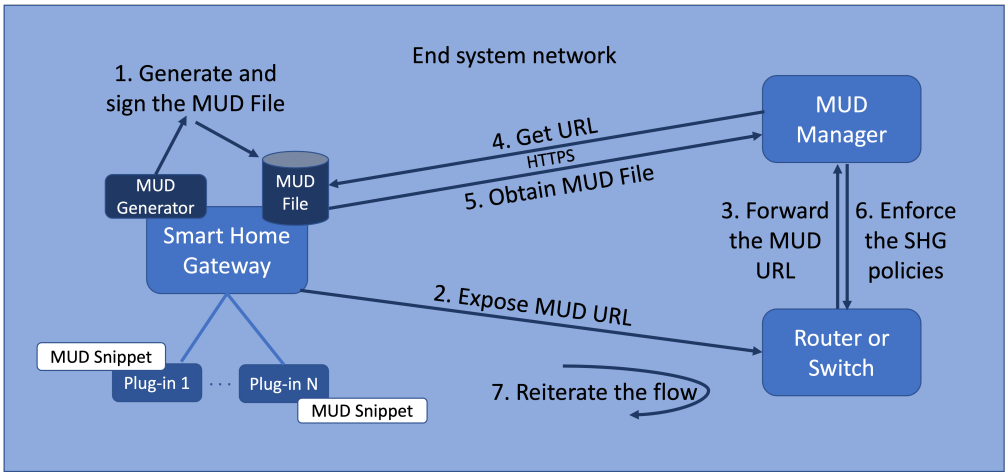

Figure 3.3: Extended MUD-architecture

3.2.2 MUD Aggregator

The workflow of *MUD Aggregator* was described in 3.2.1. This section further describes the software components of this integration, practically analyzing how it works and how it is built:

- The `__init__` function initializes the *MUD Aggregator* object. This initialization is based on some configuration parameters. Since Home Assistant has different versions, some folder paths may be different.

- The `generate_mud_file` method is in charge of generating the gateway-level MUD file. The root of this MUD file is the MUD draft (i.e., MUD skeleton), which contains all the header fields of a valid MUD file (e.g., `mud-version`, `mud-url`, `cache-validity`, `is-supported`, `systeminfo`, `mfg-name`, `last-`

`update`, `documentation`, and `model-name`), to which all the rules of the MUD snippets are added.

  - If a MUD file has already been created and the new one is different from the previous one, the timestamp inside `last-update` is set to the new value and the MUD file is signed again.

- The function `_add_mud_rules` adds the Access Control Lists (ACLs), which are the containers of the policies, to the gateway-level MUD file.

- The methods `_add_rules_from_manifest` and `_add_rules_from_folders` are in charge of searching and retrieving the MUD snippets. The former reads the files `manifest.json` of every integration and looks if the presence of the snippet is declared. The latter scans all custom and standard component directories anyway.

- All rules found in the MUD snippets are added to the MUD draft using the `_add_rules_to_draft` function. This method uses a couple of functions (e.g., `_add_policies_if_not_exist` and `_add_acls_if_not_exist`) to check if the policies or ACLs to be added do not already exist in the MUD file.

- Finally, the gateway-level MUD is signed. The last step is to expose it. The function responsible for this is `expose_mud_file` (here the method of the exposure is selected).

3.2.3 Limitations

Currently the merging of different MUD snippets by the integration *MUD Aggregator* has some limitations, which I addressed in this thesis.

First, because a MUD snippet is written by a developer, manually or with the help of a tool (e.g., MUD Maker [30]), it may contain syntax or semantic errors. A badly written MUD snippet is a JSON file that is incorrectly formatted or has a key or value that is not allowed by the MUD standard. If a MUD file contains errors, there could be problems enforcing the gateway-level MUD file, so it is essential to validate the syntax and semantics of the file.

Second, some Access Control Entries (ACEs) might overlap, that is, they might refer to the same endpoints. There could be two scenarios for this:

- The action specified by the ACEs may be the same. In this case, it is possible to optimize the rules and merge them into one.

- The action specified by the ACEs could be different. Here, instead, it is necessary to resolve this ambiguity and decide which action is chosen.

In Chapter 4, we describe how we addressed these three limitations, while Chapter 5 presents the results of the experiments conducted on the proposed approach.

Chapter 4

Proposed Approach

This chapter presents the proposed approach to address the three main problems that this thesis aims to solve: the validation of the syntax and semantics of MUD files and MUD snippets, the optimization between overlapping rules with the same action, and the conflict resolution between overlapping rules with different actions. For the first problem, `mud_json_validator.py` has been developed: it is responsible for this and has been incorporated into the *MUD Aggregator* integration. Instead, for the second and third problems, the *MUD Aggregator* integration itself has been modified and improved.

4.1 Validation

Since the purpose of this thesis is to create a unique MUD file combining a series of MUD snippets, a fundamental step is to check the syntax, the semantics, and the content of each piece of code. The validation is applied to both MUD files and MUD snippets, and depending on the type, a different type of check is done. This control is performed at three different stages (see Figure 4.1):

- When the MUD draft (i.e., MUD skeleton) is loaded.
  - It is the root of the generated MUD file and it is fundamental that is written correctly: if there is an error, the gateway-level MUD file isn't generated and the process is stopped.
- When a MUD snippet is found.
  - Here we have implemented two different strategies in case of an error, the wrong MUD snipped can be skipped (and the gateway-level MUD file is generated without including it), or the entire process is stopped (no MUD

file is exposed). This behavior is configurable through a parameter (`0` for the first option, `1` for the second).

- When the gateway-level MUD file is generated.
  - To control if the integration has correctly merged the various MUD snippets together with the MUD skeleton.

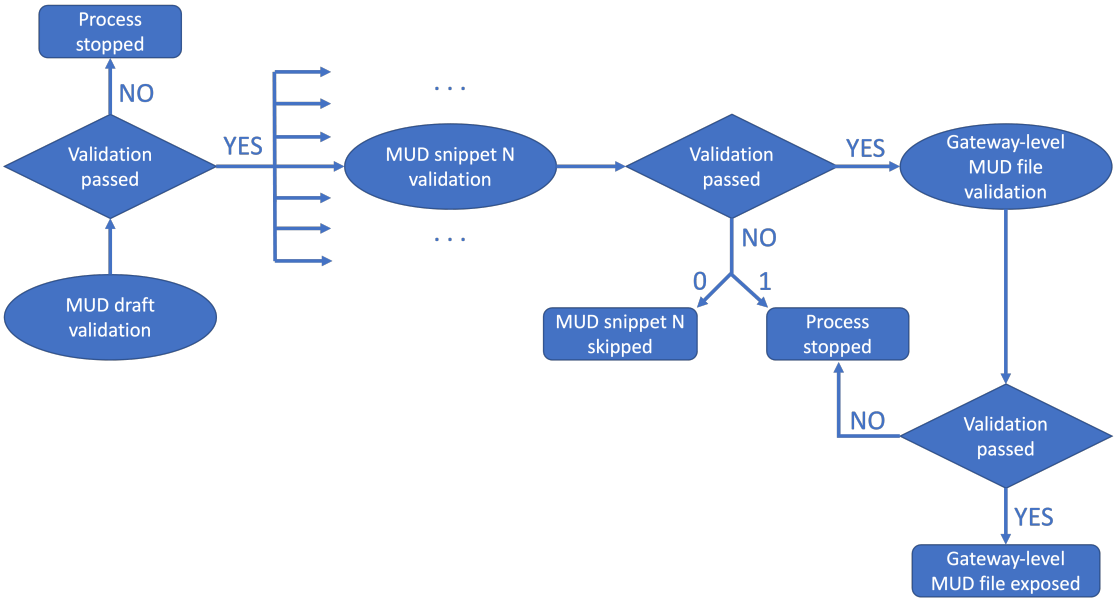

Figure 4.1: Flowchart of the validation process

In our implementation, the script responsible for the validation process is `mud_json_validator.py`, which contains a JSON schema that is used to analyze the structure of a given MUD snippet or MUD file: the only thing to do is to set the parameter `fileType` to `snippet` for the first option, to `file` for the second. In this way, it is possible to perform validation at all three mentioned steps using the same piece of code.

Most of the schema is the same for both: a MUD snippet is, in fact, a subset of a MUD file and it contains only the policies to be applied from the plug-in and to the plug-in. The `mudFileSchema`, in addition to all the controls performed by the `mudSnippetSchema`, checks the presence and analyze the values of the following properties of the `ietf-mud:mud` object, which are the headers of a MUD File: `mud-version`, `mud-url`, `last-update`, `is-supported` (that are also mandatory), `mud-signature`, `cache-validity`, `sys-teminfo`, `mfg-name`, `documentation`, `firmware-rev`, `software-rev`, `model-name` and `extensions`.

These fields are specified in the `IETF-MUD YANG Module` contained in the RFC 8520 [4], that describes the valid structure of a MUD file, by specifying which keys are valid and which keys are mandatory or optional. In addition, the `IETF-MUD YANG Module` specifies other rules and constraints, all of which are replicated in our JSON schema.

It is also possible to decide if the schema must exclude the presence of additional keys if the parameter `allowedAdditionalProperties` is `false`: this type of control, besides this function, is very useful to detect typography errors because, if a property is badly written, it results as an extra one, not present in the MUD schema.

The main tasks of the script `mud_json_validator.py` are:

- To check the type of each key in the MUD file or MUD snippet (see Listing 4.1): the possible values are five (array, number, string, boolean, and object).

Listing 4.1: One field for each possible value type

```
    "ietf-acldns:dst-dnsname":{
        "type":"string"
    },
    "protocol":{
        "type":"number"
    },
    "access-list":{
        "type":"array"
    },
    "is-supported":{
        "type":"boolean"
    },
    "ietf-mud:mud":{
        "type":"object"
    }

```

- To specify that some keys must be mandatory (see Listing 4.2): this is done through the attribute `required`.

Listing 4.2: Example of some fields that must be present in the MUD file

```
    "required": ["name", "matches", "actions"]

```

- To specify that some keys (e.g., the keys `operator`, `ethertype`, `forwarding`, `logging`, and `type`) can assume only certain values, specified in an array called `enum` (see Listing 4.3).

Listing 4.3: Defining operator `enum`

```
    "operator": {
        "type": "string",
        "enum": ["eq","lte","gte","neq","range"]
    }

```

- For some keys of type `number` (e.g., `port`, `lower-port`, `upper-port`, `port`, and `cache-validity`) it is possible to specify a range of possible values using the attributes `minimum` and `maximum` (see Listing 4.4).

Listing 4.4: Defining port field with a specified range

```
    "port": {
        "type": "number",
        "minimum": 0,"maximum": 65535
    }

```

- For the `extension` array, to check its size and to give a minimum and a maximum length through the attributes `minItems` and `maxItems` (see Listing 4.5).

Listing 4.5: Defining destination-mac-address through regex

```
    "type": "array",
    "items": {"type": "string"},
    "minItems": 1, "maxItems": 40

```

- Using regular expressions (regex), to verify if `destination-ipv4-network` and `source-ipv4-network` are IPv4 addresses, if `destination-ipv6-network` and `source-ipv6-network` are IPv6 addresses, if `mud-url`, `mud-signature`, and `documentation` are URLs, and if the key `destination-mac-address` is a MAC [31] address (see Listing 4.6).

Listing 4.6: Example of control through regex

```
    "destination-mac-address": {
        "type": "string",
        "pattern":
            "^[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}$"
    }

```

- As regards ports, to control the value of the key `operator`. If it is `range` the script verifies the presence of the keys `upper-port` and `lower-port` and excludes the presence of the key `port`, otherwise, if it is `eq` (equal), `lte` (lower than or equal), `gte` (greater than or equal) or `neq` (not equal) it verifies the opposite check.

- To control if the name of every Access Control List (ACL) contained in the objects `from-device-policy` and `to-device-policy` is then present and is the same to the name of one of the aces (Access Control Entries).

During the implementation of this schema, I came across a tool named `mud.yang.go` [32] that has the task of analyzing and validating MUD files against the YANG specification for MUDs. This tool has so the same purpose of the JSON schema presented in this work, but it presents the problem that it performs some wrong controls, leading to unexpected outcomes, e.g. it doesn't accept `range` as valid value for enum field `Operator`. In addition, since `mud_json_validator.py` is written in Python [33] and `mud.yang.go` is written in Go [34], the former guarantees better compatibility with the *MUD Aggregator* integration and better performance in terms of speed.

Then, the other difference between `mud.yang.go` and the JSON schema is that the former only allows the latest version of the MUD RFC, the latter also accepts earlier versions [35], since a significant number of MUD files are compatible with them, so:

- I allowed the possibility of naming the part of the MUD file where there are the Access Control Lists (ACLs) in two different and equivalent ways: `ietf-access-control-list:acls` and `ietf-access-control-list:access-lists`, `mud.yang.go` allowed only the former.

- I allowed the possibility of naming the ethernet Access List (ACL) type in two different and equivalent ways: `eth-acl-type` and `ethernet-acl-type`, the tool allowed only `eth-acl-type`.

- I allowed the possibility of writing the `ethertype` in decimal and in hexadecimal, the tool allowed only decimal values.

4.2 Rule Relations

In the context of firewall, as mentioned in Section 2.1, the relations among rules can assume five different forms and they are represented using Euler-Venn [36] diagrams from figure 4.2 to 4.6:

- They are "exactly matching" (EM): every field of R_x is the same of R_y (see Figure 4.2).

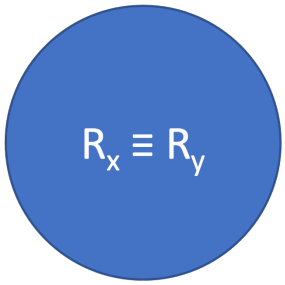

Figure 4.2: Rules "exactly matching"

- They are "inclusively matching" (IM): they do not exactly match and every field in R_x is a subset or equal to the corresponding field in R_y (see Figure 4.3).

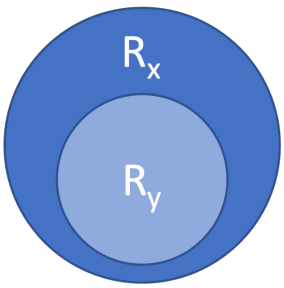

Figure 4.3: Rules "inclusively matching"

- They are "partially matching" (PM): at least one field in R_x is a subset or a super-set or equal to the corresponding field in R_y and at least one field in R_x is not a subset and not a super-set and not equal to the corresponding field in R_y (see Figure 4.4).

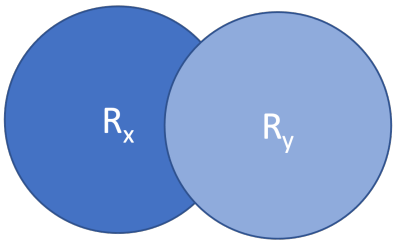

Figure 4.4: Rules "partially matching"

- They are "completely disjoint" (CD): every field of R_x is different to the one in R_y (see Figure 4.5).

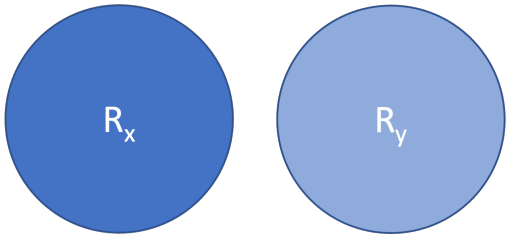

Figure 4.5: Rules "completely disjoint"

- They are "correlated" (C): some fields in R_x are subsets or equal to the corresponding fields in R_y, and the rest of the fields in R_x are super-sets of the corresponding fields in R_y (see Figure 4.6).

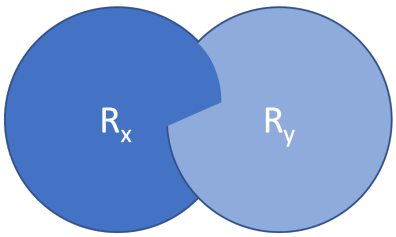

Figure 4.6: Rules "correlated"

Talking about MUD and reasoning about possible optimizations and conflicts, only the first three relations can happen. If they are "completely disjoint", since that every field is different, there isn't the necessity to optimize or to resolve conflicts. The last option instead can happen only when we specify an address range and a port range, but since in the MUD rules is not possible to specify an address range, this case is not taken into account.

Since that the MUD rules are usually translated in firewall rules and given the theorem 1 of [11] that says, "Any two k-tuple filters in a firewall policy are related by one and only one of the defined relations", we can affirm that only one of the previously described option can occur, regardless of the order of the couple. This is an important condition to guarantee that the algorithm always acts in the same way.

4.2.1 Optimizations

The optimization is evident and relevant in two cases:

- When the policies parameters are the same, i.e., both rules specify the same IP [10] version (e.g., IPv4) with the same endpoint (an IP address or a URL) and both specify the same transport protocol [37] (e.g., TCP [38]) with a specified port value.

- When both rules specify the same IP version (e.g., IPv4) with the same endpoint, but only one rule specifies the transport protocol (e.g., UDP [39]) with a specified port value.

In the second case, we have two rules "inclusively matching": the address is the same, but the transport protocol for R1 (the one in blue) is "ANY" with port "ANY", for R2 (the one in red) is "TCP" with ports from X to Y (see Figure 4.7). It is a demonstrable fact that R1 is a super-set (see Section 4.2) of R2 and that the packets that match R2 will always match also R1, so in this case R2 can be removed (see Figure 4.8).

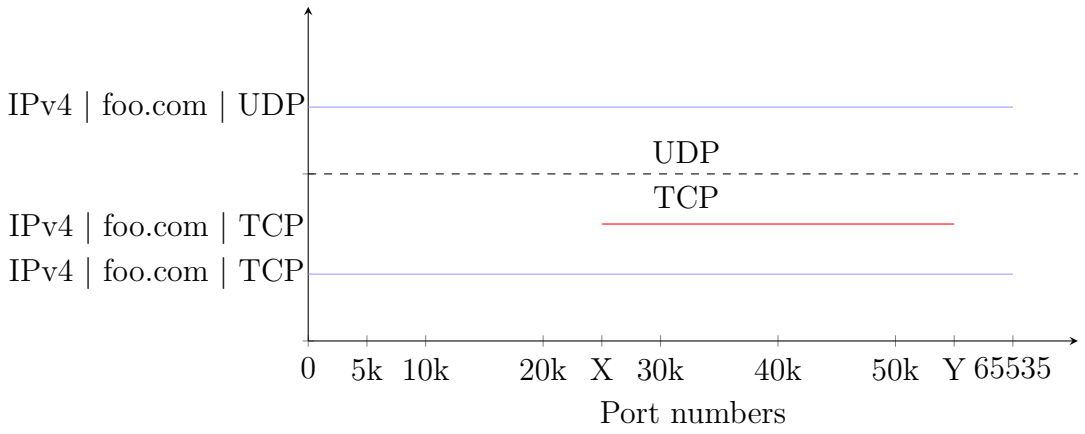

Figure 4.7: Rules "inclusively matching"

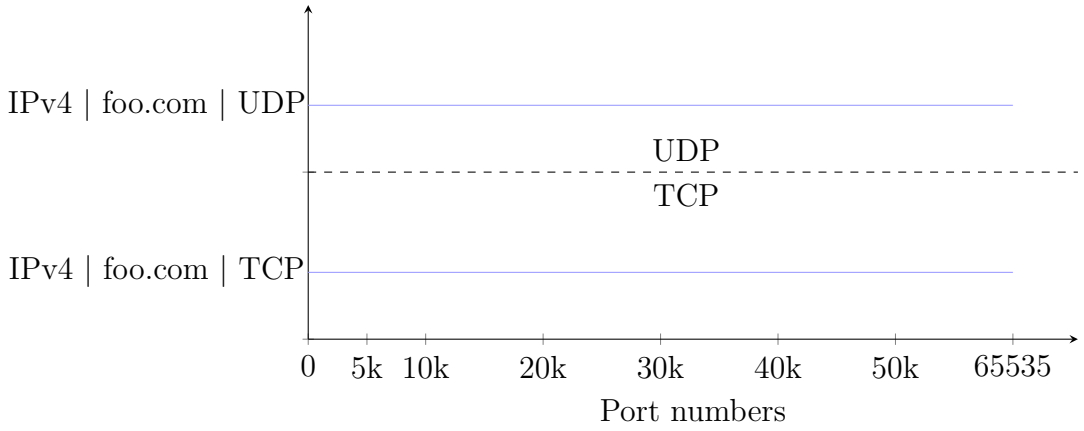

Figure 4.8: The smallest rule is removed

In the first case, that is when both rules specify the same IP version with the same endpoint and same transport protocol with a specified port value, we can have two rules "exactly matching", "inclusively matching" or "partially matching".

- In the first instance, all the parameters are equal and all the packets that match R1 (the one in red) they match also R2 (the one in blue) and vice versa: the IP protocols, the endpoints, the transport protocols and the port numbers are all the same (see Figure 4.9). So, one rule can be removed (see Figure 4.10).

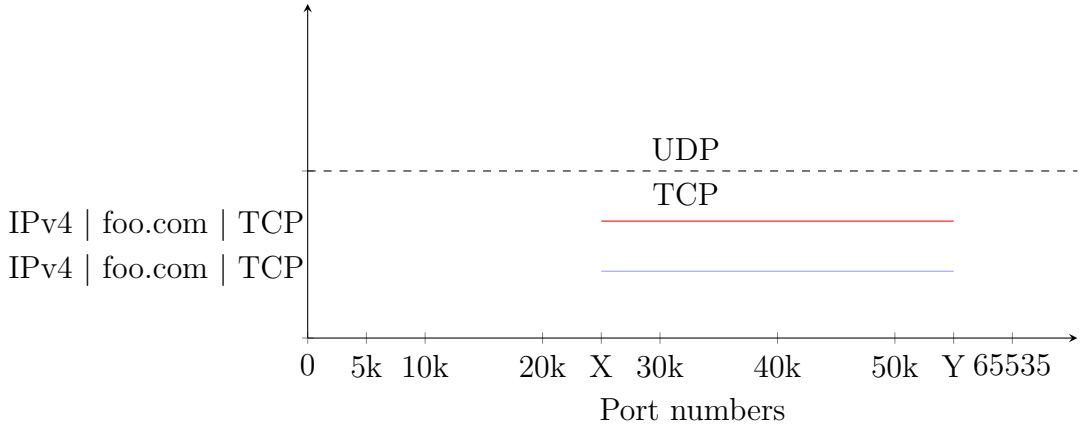

Figure 4.9: Rules "exactly matching"

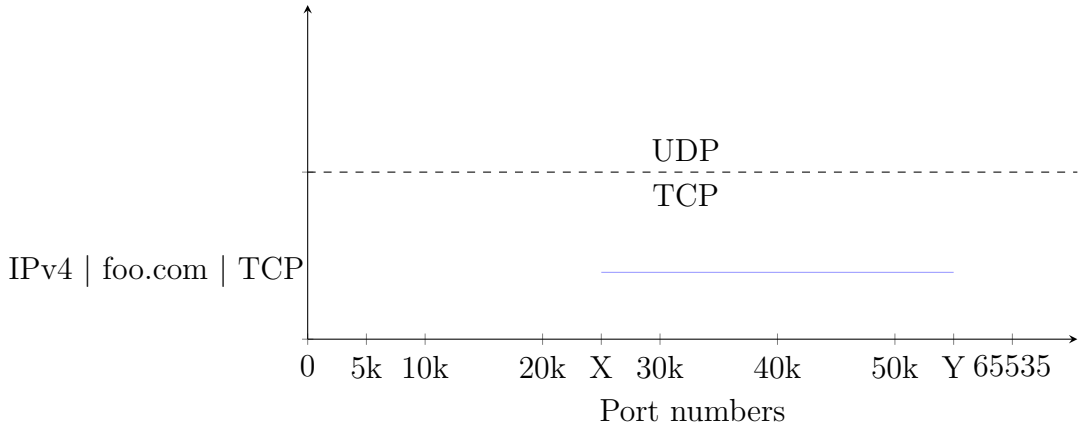

Figure 4.10: One rule is removed

- In the other instances, we can perform an optimization only if the port numbers are related: it means that some ports need to be in common, because in a MUD policy it is possible to specify only one interval of ports and it is not correct to merge different disjoint ranges in a unique rule.

  Assuming that R1 and R2 have the same endpoint, the same transport protocol, and that R1 (the one in red) covers the ports from W to Y and R2 (the one in blue) from X to Z, with W<X<Y<Z, we can demonstrate that some packets match only R1, some packets only R2, and some packets both rules (see Figure 4.11 and Listing 4.7). Given that the decision to take is the same, it is possible to merge R1 and R2 in a unique rule, named R3 (the one in green) that has the same endpoint, the same transport protocol, and that covers the ports from W to Z. All and only the packets belonging to R1 and R2 will match this rule (see Figure 4.12 and Listing 4.8).

Now it's time to go into more detail about how the algorithm works and what kinds of operations it performs.

The MUD snippets of the integrations, standard or custom, are retrieved and analyzed one by one: each Access List (ACL) found is taken and appended to the MUD draft (i.e., MUD skeleton), the starting point of the gateway-level MUD file.

This is where the optimization process begins: starting with the second Access Control Entry (ACE), each ACE is compared to all previous ACEs. For each comparison, if the two rules don't match "exactly", "inclusively", or "partially", it means that no optimization could be performed, so the iteration continues: when all the rules have been compared and none of them matches any of these three

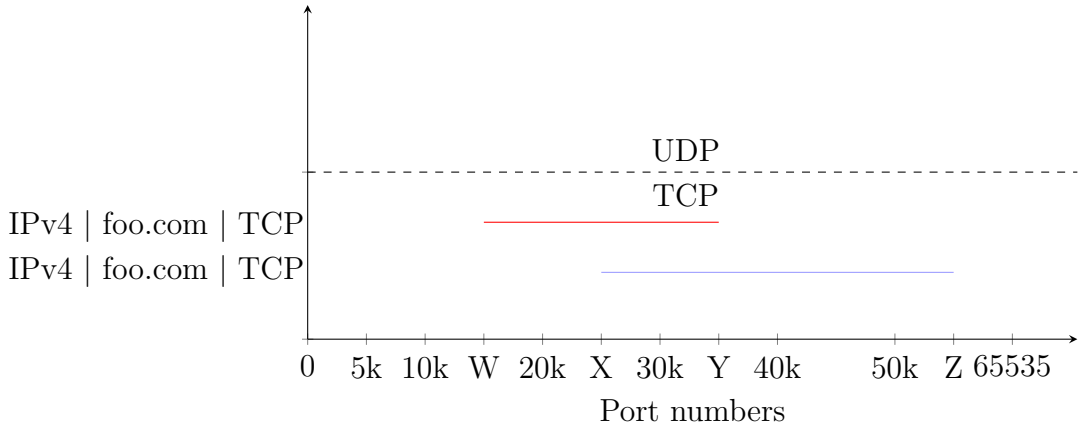

Figure 4.11: Rules "partially matching"

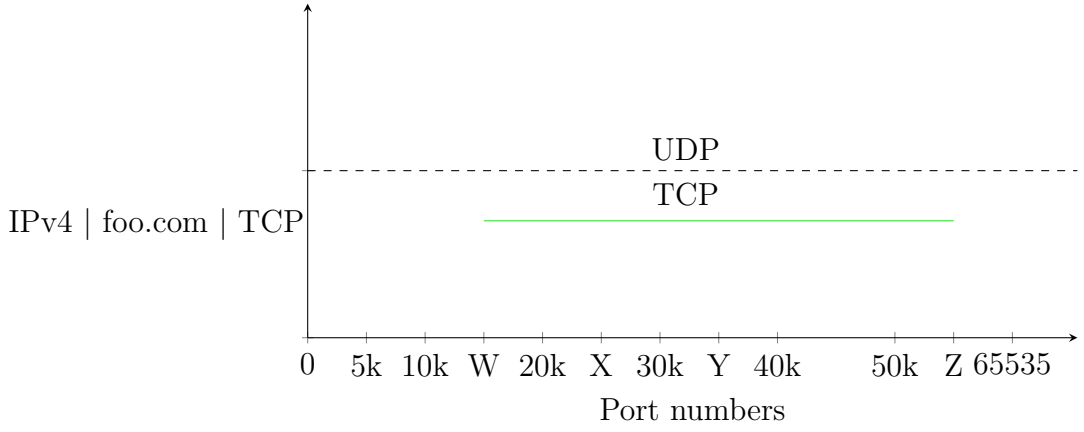

Figure 4.12: A unique rule remains

relationships, the ACE in question is preserved and control passes to the next ACE.

Instead, whenever there is an opportunity to work on the two rules, as shown in Figures 4.7, 4.9, and 4.11, the algorithm performs the optimization process: as can be seen in all three figures, when there is a matching relation, only one ACE is kept. The conserved ACE may be the original ACE if the relation is "exactly matching" or "inclusively matching", or it may be modified if the relation is "partially matching". In the first two cases, the remaining rule includes all ports of the other rule, so it remains intact. Instead, in the latter case (see Listing 4.7), we have to create a new ACE that includes the ports of both ACEs: the strategy that I have implemented consists in modifying the preserved ACE by including all the necessary ports and reporting it in the name of the ACE with the prefix `opt-` (see Listing 4.8).

Listing 4.7: Example of two ACEs "partially matching"

```
{
    "name":"from-ipv4-test-0",
    "matches":{
        "ipv4":{
            "protocol":6,
            "ietf-acldns:dst-dnsname":"foo.com"
        },
        "tcp":{
            "destination-port":{
                "operator":"range",
                "lower-port":15000,
                "upper-port":35000
            }
        }
    },
    "actions":{
        "forwarding":"accept"
    }
},
{
    "name":"from-ipv4-test-1",
    "matches":{
        "ipv4":{
            "protocol":6,
            "ietf-acldns:dst-dnsname":"foo.com"
        },
        "tcp":{
            "destination-port":{
                "operator":"range",
                "lower-port":25000,
                "upper-port":55000
            }
        }
    },
    "actions":{
        "forwarding":"accept"
    }
}

```

Listing 4.8: Example of two ACEs "partially matching" optimized

```
{
    "name":"opt-from-ipv4-test-0",
    "matches":{
        "ipv4":{
            "protocol":6,
            "ietf-acldns:dst-dnsname":"foo.com"
        },
        "tcp":{
            "destination-port":{
                "operator":"range",
                "lower-port":15000,
                "upper-port":55000
            }
        }
    },
    "actions":{
        "forwarding":"accept"
    }

```

The discarded ACE could be the ACE previously inserted or the ACE currently being compared with the old ACEs. In order not to change the behavior of the algorithm depending on whether the first or the second ACE is kept, after each optimization the process is restarted and all ACEs are compared again.

4.2.2 Conflicts

After having optimized all the rules of the gateway-level MUD file, next step to perform is to resolve conflicts. Conflicts can happen when one of the first three conditions in Section 4.2 is verified and the decision to take (allow or deny) is different.

A crucial point is to ensure the determinism of the output, so that in any order the rules are processed the generated MUD file will be the same. In my algorithm, this is done sorting the rules, according to the port range, before starting the conflict resolution process. The user can specify four different strategies:

- Allow Takes Precedence (ATP): in case of contradicting actions that are simultaneously activated the algorithm enforces the Allow rule over the Deny one.
- Deny Takes Precedence (DTP): in case of contradicting actions that are

simultaneously activated the algorithm enforces the Deny rule over the Allow one.

- Most Specific Takes Precedence (MSTP): in case two conflicting rules are applied, the most specific rule is the one that takes precedence.
- Least Specific Takes Precedence (LSTP): in case two conflicting rules are applied, the less specific rule is the one that takes precedence.

In the first three cases, the policies are sorted from the ones with the smaller port range to the ones with the highest range. For the "Least Specific Takes Precedence" option the order is reversed because the relevance is given to the widest rules. Every time a conflict is resolved, the rules are ordered again and the conflict resolution process restarts. When the process is complete and at least one conflict has been resolved, the algorithm repeats the optimization and conflict resolution until no more conflicts are found (see Figure 4.13).

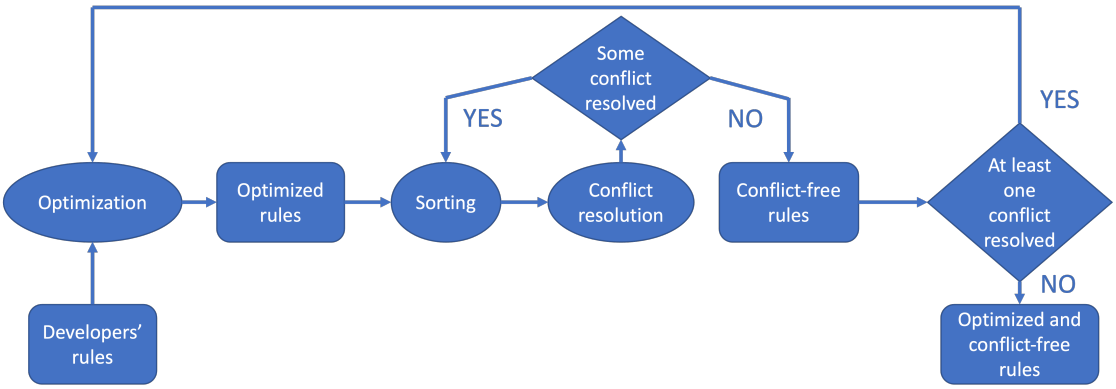

Figure 4.13: Flowchart of rules anomalies resolution

When a conflict is found, the resolution is applied only to the part of the rule where there is an overlap. The rule modified it is now recognizable through the prefix `conf-`. For example, if two rules are "inclusively matching", the rule that represents the super set is divided in two or three parts. The Figure 4.14 and the Listing 4.9 illustrate an example of a potential conflict.

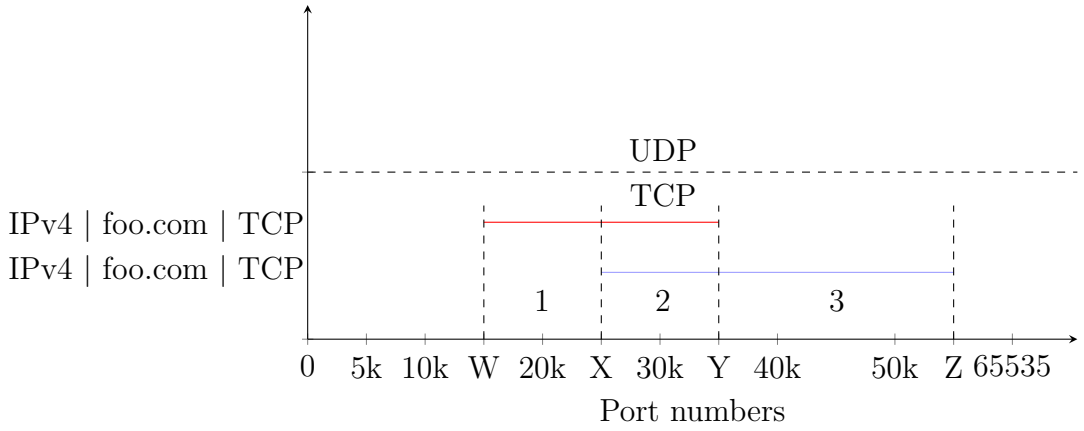

Figure 4.14: Conflicted rules "partially matching"

Listing 4.9: Conflicted rules "partially matching"

```
1    {
2       "name":"from-ipv4-test-0",
3       "matches":{
4          "ipv4":{
5             "protocol":6,
6             "ietf-acldns:dst-dnsname":"foo.com"
7          },
8          "tcp":{
9             "destination-port":{
10               "operator":"range",
11               "lower-port":15000,
12               "upper-port":35000
13            }
14         }
15      },
16      "actions":{
17         "forwarding":"accept"
18      }
19   },
20   {
21      "name":"from-ipv4-test-1",
22      "matches":{
23         "ipv4":{
24            "protocol":6,
25            "ietf-acldns:dst-dnsname":"foo.com"
```

```
26              },
27              "tcp":{
28                  "destination-port":{
29                      "operator":"range",
30                      "lower-port":25000,
31                      "upper-port":55000
32                  }
33              }
34          },
35          "actions":{
36              "forwarding":"reject"
37          }
38      }
39
```

The parts of the first rule that aren't in common with the second rule remain intact (segments 1 and 3); the other part (segment 2) is subject to the conflict resolution process, depending on the option chosen by the user. The segments 1 and 3 are not changed and, assuming that the rule in red has the "accept" action and that the blue has the "deny" action, the segment 2 could become:

- If the option chosen is ATP, it becomes part of the red rule, because this rule is the permissive one (see Figure 4.15 and Listing 4.10).
- If the option chosen is DTP, it becomes part of the blue rule, because this rule is the prohibitive one (see Figure 4.16 and Listing 4.11).
- If the option chosen is MSTP, it becomes part of the red rule, because this rule is the smallest (see Figure 4.15 and Listing 4.10).
- If the option chosen is LSTP, it becomes part of the blue rule, because this rule is the widest (see Figure 4.16 and Listing 4.11).

In this way, we interact only with the parts of the rules subject to a conflict. For the other parts, as can be seen in Figures 4.15 and 4.16, the segments not involved in the conflict resolution process are not modified, and the network accessible before and after this operation is the same.

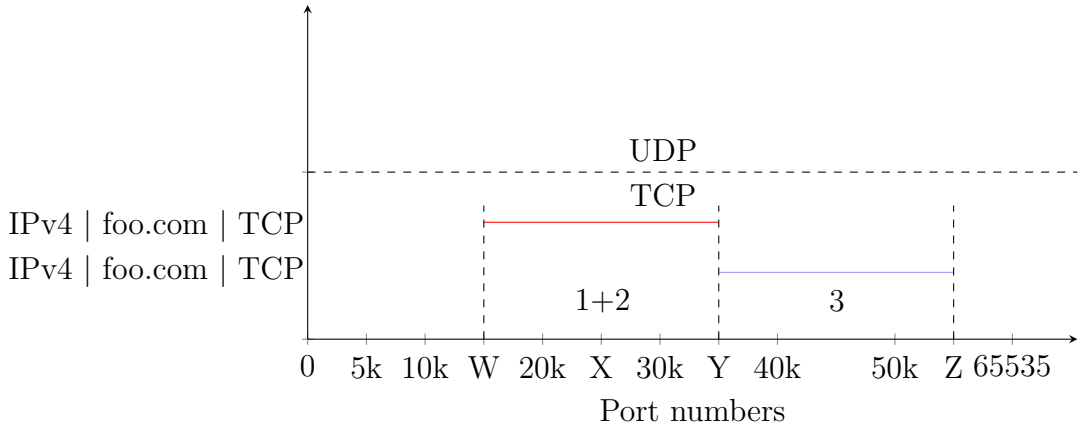

Figure 4.15: Conflict resolved for rules "partially matching" (ATP and MSTP)

Listing 4.10: Conflict resolved for rules "partially matching" (ATP and MSTP)

```
    {
        "name":"from-ipv4-test-0",
        "matches":{
            "ipv4":{
                "protocol":6,
                "ietf-acldns:dst-dnsname":"foo.com"
            },
            "tcp":{
                "destination-port":{
                    "operator":"range",
                    "lower-port":15000,
                    "upper-port":35000
                }
            }
        },
        "actions":{
            "forwarding":"accept"
        }
    },
    {
        "name":"conf-from-ipv4-test-1",
        "matches":{
            "ipv4":{
                "protocol":6,
                "ietf-acldns:dst-dnsname":"foo.com"
```

```
            },
            "tcp":{
               "destination-port":{
                  "operator":"range",
                  "lower-port":35000,
                  "upper-port":55000
               }
            }
         },
         "actions":{
            "forwarding":"reject"
         }
      }

```

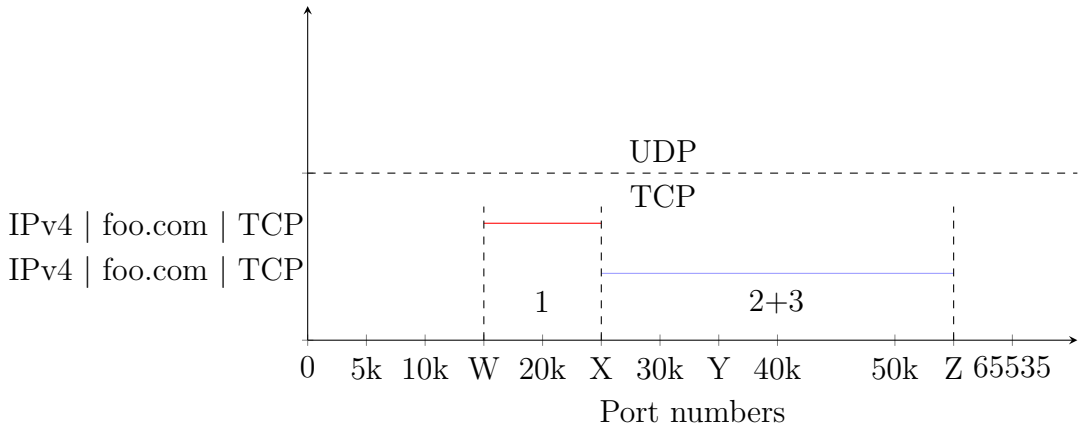

Figure 4.16: Conflict resolved for rules “partially matching” (DTP and LSTP)

Listing 4.11: Conflict resolved for rules “partially matching” (DTP and LSTP)

```
1  {
2     "name":"conf-from-ipv4-test-0",
3     "matches":{
4        "ipv4":{
5           "protocol":6,
6           "ietf-acldns:dst-dnsname":"foo.com"
7        },
8        "tcp":{
9           "destination-port":{
10             "operator":"range",
```

```
                "lower-port":15000,
                "upper-port":25000
            }
        }
    },
    "actions":{
        "forwarding":"accept"
    }
},
{
    "name":"from-ipv4-test-1",
    "matches":{
        "ipv4":{
            "protocol":6,
            "ietf-acldns:dst-dnsname":"foo.com"
        },
        "tcp":{
            "destination-port":{
                "operator":"range",
                "lower-port":25000,
                "upper-port":55000
            }
        }
    },
    "actions":{
        "forwarding":"reject"
    }
}

```

It is important to note that there is only one case in which the number of rules can increase during the conflict resolution process. This happens when two rules are "inclusively matching" and the winning rule is the smaller one (see Figure 4.17). In this case, the largest rule is split into two parts (the ones not involved in the conflict), increasing the number of rules (see Figure 4.18). In all other situations, the number of rules remains the same.

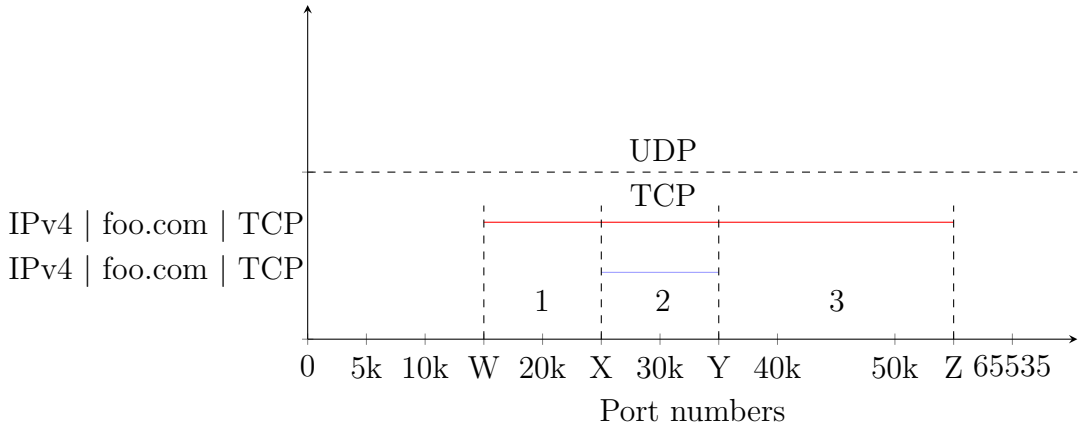

Figure 4.17: Conflicted rules "inclusively matching"

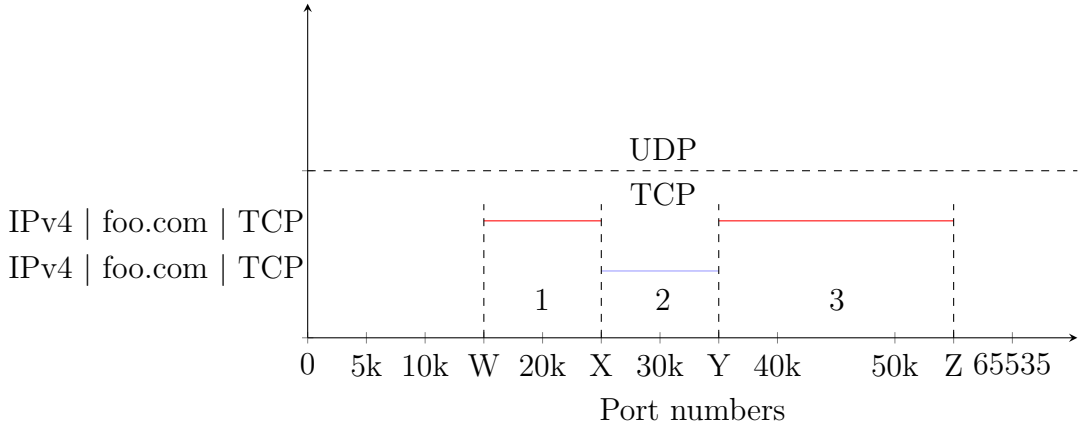

Figure 4.18: Conflict resolved for rules "inclusively matching" (DTP and LSTP)

Chapter 5

Experimental Results

This chapter presents the results of the analysis and of the management of MUD (Manufacturer Usage Description) files. The syntactic validation of MUD snippets, and the complex task of optimizing and resolving conflicts between rules, are explored in this study. The purpose of this chapter is to demonstrate the improvements that have resulted from this work.

Our approaches were carefully designed and executed to validate the solution presented, ensuring practicality and effectiveness.

In the forthcoming sections, we shall examine our experimental methodology, the obtained results, and the potential implications of our findings, presenting a comprehensive overview of the contributions made to the MUD ecosystem.

5.1 Experimental setup

In this section, the experimental setup configured in the laboratory of "Politecnico di Torino" is described. The setup consists of two Raspberry Pi 3 Model B v1.2[1]:

- The first is equipped with OpenWRT (version 17.01.6) and osMUD; specifically a customized version created starting from version 0.2.0: it has the ability to handle a larger number of MUD snippets and to accept the range port option in MUD files.
- The second board is equipped with Home Assistant; specifically with Home Assistant OS 10.3, Home Assistant Core 2023.7.5 and Home Assistant Supervisor 2023.07.3.

[1] `https://www.raspberrypi.com/products/raspberry-pi-3-model-b/`, last visited on November 27th, 2023.

5.2 Validation

A crucial point of this thesis is the validation of the syntax and structure of a MUD file or MUD snippet. To prove my work, I have analyzed a large set of MUD files retrieved on the Internet and I have compared the results of my script and the results obtained with the tool `mud.yang.go`. These MUD files (more than 60) have been taken from two different sources:

- The first is the repository[2] of MUD Visualizer [40], where there is a group of files generated by MUDgee [41], a tool that can generate MUD files using PCAP [42], which is an interface for capturing network traffic.
- The second is IoTOPIA[3], a platform where device manufacturers can publish, in a unique location, the MUD file associated with their products.

The comparison results are available in Table 5.1 and Table 5.2. The former includes MUD files obtained from the first source, MUD Visualizer, while the latter includes MUD files obtained from IoTOPIA.

In order to focus on different types of errors, I have modified the MUD files, by inserting and correcting the keys `ietf-access-control-list:access-lists`, `ethernet-acl-type` and the ethertype values: as mentioned in Section 4.1, `mud .yang.go` only allows the latest version of the MUD RFC, instead `mud_json_validator.py` accepts previous versions as well, since most of the retrieved MUD files are compliant with them.

Since the shutdown of IoTOPIA on August 8th, 2023, the MUD files coming from this platform have been stored in this repository[4], along with the MUD files coming from MUD visualizer, modified as described above.

Table 5.1: Comparison results between `mud.yang.go` and `mud_json_validator.py`: IoTOPIA files

IoTOPIA	mud_json_validator.py	mud.yang.go
withingsbaby monitorMud_2020 1203151125.json	Valid	Valid

[2] `https://github.com/iot-onboarding/mud-visualizer/tree/master/sample_mud_files/mudgee_mudfiles`, last visited on November 27th, 2023.

[3] `https://mudfileservice.globalplatform.org`, last visited on August 2nd, 2023.

[4] `https://github.com/emalinty/mud_files`, last visited on November 27th, 2023.

IoTOPIA	**mud_json_validator.py**	**mud.yang.go**
fe-same manufacturer-from2_2020 1203150834.json	NOT valid: invalid URL + "mud-75085-v4to" NOT present in target node	Valid BUT "mud-75085-v4to" NOT present in target node
F7C029_ 20201203150644.json	Valid	Valid
NetatmoWeatherStation Mud_20201203150842.json	Valid	Valid
nestsmoke sensor_2020 1203150910.json	Valid	Valid
fe-same manufacturer-from2_202012 03151059.json	NOT valid: invalid URL + "mud-75085-v4to" NOT present in target node	Valid BUT "mud-75085-v4to" NOT present in target node
is_imilab_ cameraMud_20 210328050145.json	Valid	Valid
Securelock_2020 1207153351.json	NOT valid: invalid URL + "my-controller" isn't an array	NOT valid: "my-controller" isn't an array
fe-localnetwork_ 20201203150917.json	Valid	Valid
ciscopi21_2020 1203150829.json	NOT valid: invalid URL	Valid
mud_crm_ver1_20 221110065713.json	NOT valid: invalid URL	Valid
lightbulb2020_202 01203151105.json	Valid	Valid
amazonEchoMud_2 0201203150838.json	Valid	Valid
lightbulb2020_20 201102124202.json	Valid	Valid
cr-5b.json	NOT valid: invalid URL	Valid
fjdskfsdlfkjd_20 210121113559.json	NOT valid: invalid URL	Valid

IoTOPIA	**mud_json_validator.py**	**mud.yang.go**
WSP080_ 20201203150703.json	Valid	Valid
ringdoorbellMud_20 210227203739.json	Valid	Valid
withingsbabymonitorMud_ 20210322143827.json	Valid	Valid
amazonEchoMud_ 20201203151116.json	Valid	Valid
cr-5b_20201218205035.json	NOT valid: invalid URL	Valid
cr-5b_20201203151111.json	NOT valid: invalid URL	Valid
ACME1_ 20201203150723.json	NOT valid: invalid URL	Valid
WSP080_ 20201218203803.json	NOT valid: invalid URL	Valid
ciscopi2_ 20201216162818.json	NOT valid: invalid URL	Valid
ch_imilab_cameraMud_ 20210328050015.json	Valid	Valid
withingsbabymonitorMud_ 20201203151134.json	Valid	Valid
repeater2b_2020 1218202321.json	Valid	Valid
cr-5b_20201203151049.json	NOT valid: invalid URL	Valid
HueBulbMud_2021 0301152627.json	Valid	Valid
lightbulb2020_20201203151 152_20230221143921.json	Valid	Valid
coffee_multiple_20 221108110209.json	Valid	Valid
repeater2b_2020 1218202339.json	Valid	Valid
withingsbabymonitorMud_ 20201203150905.json	Valid	Valid
withingssleepsensorMud_ 20210227204045.json	Valid	Valid

IoTOPIA	**mud_json_validator.py**	**mud.yang.go**
cr-5b_20201203151130.json	NOT valid: invalid URL	Valid
WSP080_ 20201203150540.json	NOT valid: invalid URL	Valid
mud_crm_ver1_20 221110070753.json	NOT valid: invalid URL	Valid
Securelock_2020 1203150815.json	NOT valid: invalid URL	Valid

Table 5.2: Comparison results between `mud.yang.go` and `mud_json_validator.py`: MUDGee files

MUDGee	**mud_json_validator.py**	**mud.yang.go**
hellobarbieMud.json	Valid	Valid
pixstarphotoframeMud.json	Valid	Valid
tplinkcameraMud.json	Valid	NOT valid: operator "range" not valid
withingssleepsensorMud.json	Valid	Valid
NetatmoCameraMud.json	Valid	Valid
lifxbulbMud.json	Valid	Valid
hpprinterMud.json	Valid	NOT valid: source and destination ipv6 network do not match the pattern
NetatmoWeather StationMud.json	Valid	Valid
chromecastUltraMud.json	Valid	Valid
dropcamMud.json	Valid	Valid
blipcareBPmeterMud.json	Valid	Valid
HueBulbMud.json	Valid	Valid
withingsbaby monitorMud.json	Valid	Valid
wemomotionMud.json	Valid	Valid
wemoswitchMud.json	Valid	Valid
ringdoorbellMud.json	Valid	Valid
awairAirQualityMud.json	Valid	Valid

MUDGee	**mud_json_validator.py**	**mud.yang.go**
belkincameraMud.json	Valid	Valid
ihomepowerplugMud.json	Valid	Valid
amazonEchoMud.json	Valid	Valid
tplinkplugMud.json	Valid	Valid
SmartThingsMud.json	Valid	Valid
nestsmokesensorMud.json	Valid	Valid
tribyspeakerMud.json	Valid	Valid
withingscardioMud.json	Valid	Valid
augustdoorbell camMud.json	NOT valid: "192.168.1.1" isn't an ipv6 address	NOT valid: "192.168.1.1" isn't an ipv6 address and source and destination ipv6 network do not match the pattern
samsungsmart camMud.json	NOT valid: problems with upper-port and lower-port with operator "eq"	NOT valid: problems with upper-port and lower-port with operator "eq"

The results are quite similar, most of the MUD files results valid and most errors are the same: the files with the same output are 46 (68%). However, according to the conducted tests, our proposed solution better analyzes 21 files (32%) and has the following pros:

- It detects badly formatted URLs, e.g., "`https://oururl/serialnumber`" or "`https://https://www.cisco.com/us/p/P-WSP080//ciscopi2.p7s`"
- It recognizes `range` as a valid value for the port numbers.
- The tool `mud.yang.go` sometimes raises an exception of "invalid address" even if an IPv6 address is correct.
- It is faster: `mud_json_validator.py` results about 90% quicker, from an average of 4.7s to an average of 0.35s.

Regarding the last point, the results of the comparison time between `mud.yang.go` and `mud_json_validator.py` are presented in Table 5.3 and in Table 5.4. It is evident that `mud_json_validator.py` is faster: it takes, more or less, one-fourth the time of `mud.yang.go`

Table 5.3: Comparison time between `mud.yang.go` and `mud_json_validator.py`: IoTOPIA files

IoTOPIA	**mud.yang.go time**	**mud_json_validator.py time**
WSP080_20201203150540.json	6.08s	0.35s
WSP080_20201203150703.json	4.27s	0.34s
withingsbabymonitorMud_ 20201203151125.json	4.95s	0.38s
fe-samemanufacturer-from2_20201203150834.json	7.78s	0.33s
nestsmokesensor_ 20201203150910.json	6.72s	0.35s
fe-samemanufacturer-from2_ 20201203151059.json	5.80s	0.34s
is_imilab_cameraMud_ 20210328050145.json	4.55s	0.34s
Securelock_ 20201207153351.json	4.49s	0.39s
fe-localnetwork_ 20201203150917.json	4.38s	0.37s
ciscopi21_20201203150829.json	6.80s	0.36s
mud_crm_ver1_ 20221110065713.json	4.45s	0.34s
lightbulb2020_ 20201203151105.json	4.33s	0.33s
amazonEchoMud_ 20201203150838.json	4.28s	0.45s
lightbulb2020_ 20201102124202.json	4.32s	0.34s
NetatmoWeatherStationMud_ 20201203150842.json	4.53s	0.36s
cr-5b.json	4.58s	0.37s
fjdskfsdlfkjd_ 20210121113559.json	4.38s	0.36s

IoTOPIA	**mud.yang.go time**	**mud_json_validator.py time**
ringdoorbellMud_ 20210227203739.json	4.65s	0.44s
withingsbabymonitorMud_ 20210322143827.json	5.87s	0.35s
F7C029_20201203150644.json	4.33s	0.35s
cr-5b_20201218205035.json	4.62s	0.34s
amazonEchoMud_ 20201203151116.json	5.60s	0.36s
cr-5b_20201203151111.json	4.63s	0.36s
ACME1_20201203150723.json	4.73s	0.35s
ch_imilab_cameraMud_ 20210328050015.json	4.85s	0.35s
WSP080_20201218203803.json	4.31s	0.36s
ciscopi2_20201216162818.json	4.60s	0.44s
repeater2b_ 20201218202321.json	4.78s	0.34s
withingsbabymonitorMud_ 20201203151134.json	4.78s	0.37s
cr-5b_20201203151049.json	5.46s	0.43s
HueBulbMud_ 20210301152627.json	5.17s	0.35s
lightbulb2020_20201203151 152_20230221143921.json	4.37s	0.34s
coffee_multiple_ 20221108110209.json	4.69s	0.38s
cr-5b_20201203151130.json	4.98s	0.43s
repeater2b_ 20201218202339.json	4.48s	0.32s
mud_crm_ver1_ 20221110070753.json	4.71s	0.33s
withingsbabymonitorMud_ 20201203150905.json	4.69s	0.36s
withingssleepsensorMud_ 20210227204045.json	4.58s	0.33s
Securelock_ 20201203150815.json	4.45s	0.40s

Table 5.4: Comparison time between `mud.yang.go` and `mud_json_validator.py`: MUDGee files

MUDGee	mud.yang.go time	mud_json_validator.py time
hellobarbieMud.json	6.08s	0.34s
pixstarphotoframeMud.json	4.69s	0.34s
tplinkcameraMud.json	4.87s	0.43s
withingssleepsensorMud.json	4.63s	0.34s
NetatmoCameraMud.json	5.81s	0.35s
lifxbulbMud.json	4.67s	0.34s
hpprinterMud.json	5.56s	0.36s
chromecastUltraMud.json	8.79s	0.33s
dropcamMud.json	9.68s	0.35s
blipcareBPmeterMud.json	4.45s	0.45s
HueBulbMud.json	9.89s	0.34s
withingsbabymonitorMud.json	4.69s	0.34s
wemomotionMud.json	5.01s	0.35s
wemoswitchMud.json	4.98s	0.33s
ringdoorbellMud.json	4.94s	0.32s
awairAirQualityMud.json	4.89s	0.46s
belkincameraMud.json	5.25s	0.35s
ihomepowerplugMud.json	4.66s	0.35s
amazonEchoMud.json	5.54s	0.34s
augustdoorbellcamMud.json	6.14s	0.43s
tplinkplugMud.json	6.02s	0.36s
SmartThingsMud.json	4.71s	0.36s
nestsmokesensorMud.json	6.65s	0.34s
tribyspeakerMud.json	6.90s	0.34s
withingscardioMud.json	4.57s	0.44s
samsungsmartcamMud.json	5.16s	0.33s
NetatmoWeather StationMud.json	4.52s	0.35s

5.3 Optimizations

In order to validate the proposed approach presented for the optimization process, I have defined four sets of MUD snippets. The MUD snippets were retrieved from the corresponding MUD files generated by the tool MUDGee. This tool generates MUD files using PCAP, an interface for capturing network traffic.

Therefore, if we remove the headers related to a complete MUD file, we can assume that they are comparable to MUD snippets written by developers and used for a potential respective integration in Home Assistant.

These sets are composed by MUD snippets derived from the MUD Visualizer repository and have four different characteristics to differentiate the possible case studies:

- "Set 1" is composed by MUD snippets coming from the same manufacturer (Withings [43]): `withingsbabymonitorMud.json`, `withingscardioMud.json`, and `withingssleepsensorMud.json`.
- "Set 2" is composed by two couples of MUD snippets coming from two different manufacturers (Netatmo [44] and TP-Link [45]): `NetatmoCameraMud.json`, `NetatmoWeatherStationMud.json`, `tplinkcameraMud.json`, and `tplink-plugMud.json`.
- "Set 3" is composed by a couple of MUD snippets coming from the same manufacturer (Wemo by Belkin [46]), and other two snippets coming from two different manufacturers (Awair [47] and Pix-Star [48]): `wemomotionMud.json`, `wemoswitchMud.json`, `awairAirQualityMud.json`, and `pixstarphotofra-meMud.json`.
- "Set 4" is composed by all the MUD snippets mentioned in the previous three sets.

All of these sets are available and accessible in this repository[5]. The time elapsed described in Table 5.5 is related to validation and optimization.

Set number	Rules pre and post optimization	Reduction	Time elapsed
Set 1	39 $\longrightarrow$ 29	26%	0.72s
Set 2	122 $\longrightarrow$ 94	23%	0.79s
Set 3	96 $\longrightarrow$ 55	43%	0.75s
Set 4	257 $\longrightarrow$ 165	36%	0.89s

Table 5.5: Optimization results

As shown in the Table 5.5, the visible result is a reduction in the number of Access Control Entries (ACEs) by around 30%, while maintaining the same allowed traffic space. In this way, it is possible to achieve a higher speed of rule enforcement. Assuming that a rule is repeated, if the analyzed packet doesn't match the first

[5] `https://github.com/emalinty/mud_files`, last visited on November 27th, 2023.

rule, it won't even match the second rule. The repetition is therefore unnecessary and a waste of time for the user who is waiting for the rules to be enforced when Home Assistant starts.

To provide an example, Listing 5.1 and Listing 5.2 contain two different overlapping rules that can be optimized. The first ACE, named `from-ipv4-withingsbabymonitor-7` and available in 5.1, is from `withingsbabymonitorMud.json`. The second instead, named `from-ipv4-withingssleepsensor-5` and available in 5.2, is from `withingssleepsensorMud.json`.

Listing 5.1: `from-ipv4-withingsbabymonitor-7`

```
{
    "name":"from-ipv4-withingsbabymonitor-7",
    "matches":{
        "ietf-mud:mud":{
            "local-networks":[
                null
            ]
        },
        "ipv4":{
            "protocol":17,
            "destination-ipv4-network":
                "224.0.0.251/32"
        },
        "udp":{
            "destination-port":{
                "operator":"eq",
                "port":5353
            }
        }
    },
    "actions":{
        "forwarding":"accept"
    }
}

```

Listing 5.2: `from-ipv4-withingssleepsensor-5`

```
{
    "name":"from-ipv4-withingssleepsensor-5",
    "matches":{
        "ietf-mud:mud":{
            "local-networks":[
                null
            ]
        },
        "ipv4":{
            "protocol":17,
            "destination-ipv4-network":
                "224.0.0.251/32"
        },
        "udp":{
            "destination-port":{
                "operator":"eq",
                "port":5353
            }
        }
    },
    "actions":{
        "forwarding":"accept"
    }
}

```

In both ACEs, permission is granted to communicate within the local network using the multicast address `224.0.0.251/32`, using Internet Protocol version 4 and using the UDP transport protocol on port 5353. We can say that the two rules are "exactly matching" (see Figure 5.1).

Since these rules are "exactly matching", it is possible to keep only one rule. This optimization will reduce the final number of rules. The allowed traffic (the packets going to `224.0.0.251/32` on port 5353 using UDP) remains intact and is still allowed after the optimization. In addition, this operation does not add any other endpoint or port that could lead to unexpected results (see Figure 5.2).

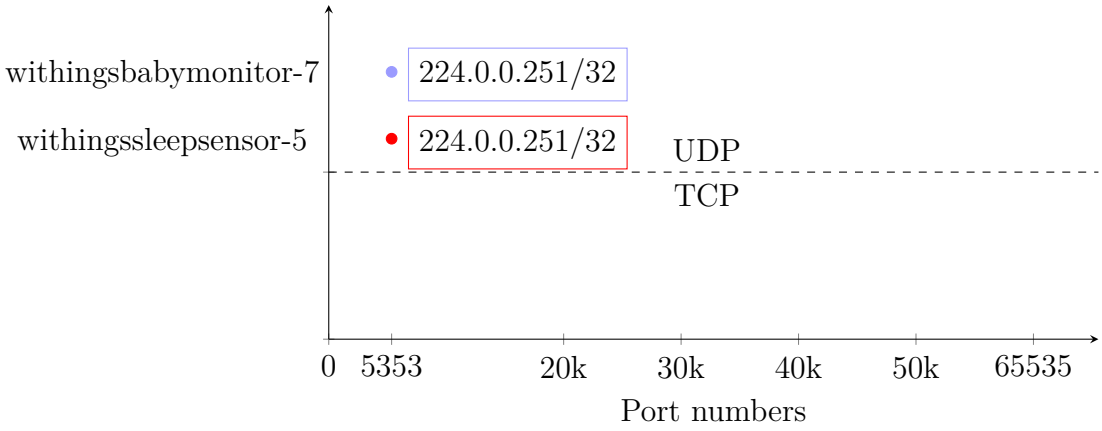

Figure 5.1: Two ACEs "exactly matching"

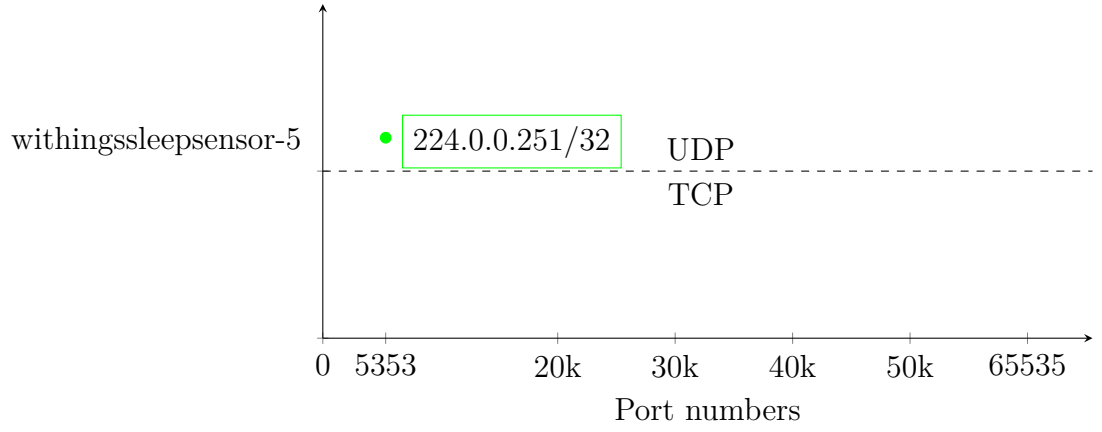

Figure 5.2: "Exactly matching" conflict resolution

5.4 Conflicts

Regarding the conflict resolution process, since the standard MUD by default blocks the traffic that is not explicitly allowed, it is rare to find Access Control Entries (ACEs) that deny the specified traffic. However, the specifications allow to set the forwarding action even to "drop" or "reject", so it is necessary to develop a strategy for handling potential conflicts.

To demonstrate the correct implementation of the conflict resolution process, I created three sets of MUD snippets, starting from MUD files retrieved from IoTOPIA and the MUD Visualizer repository (see Section 5.3). I then slightly modified these snippets to create some potential conflicts and demonstrate the correct resolution. The sets are composed as follows:

- "Set α" consists of `mod_ihomeplug.json`, `mod_blipcareMud.json`, and `mod_hellobarMud.json`.
- "Set β" consists of `mod_lifxbulb.json`, `mod_SmartThings.json`, and `mod_ringdoorbell.json`.
- "Set γ" consists of `mod_cr-5b.json`, `mod_ciscopi2.json`, and `mod_coffee_mult.json`.
- "Set δ" consists of all the MUD snippets mentioned in the previous three sets.

All of these sets are available and accessible in this repository[6]. The time elapsed described in Table 5.6 is related to validation and conflict resolution.

Set number	Number of rules pre and post conflict resolution	Overlapping rules pre and post conflict resolution	Time elapsed
Set α	30 ⟶ 33	8 ⟶ 0	0.90s
Set β	35 ⟶ 37	10 ⟶ 0	0.96s
Set γ	23 ⟶ 24	8 ⟶ 0	0.84s
Set δ	64 ⟶ 66	20 ⟶ 0	1.03s

Table 5.6: Conflict resolution results

As can be seen in Table 5.6, there are no more overlapping and conflicting rules after this process. All potential ambiguities are resolved, and each packet now matches only one rule.

To provide an example, as done in Section 5.3, Listing 5.3 and Listing 5.4 contain two different overlapping rules that present a conflict. The first ACE, named `from-ipv4-blipcarebpmeter-3` and available in Listing 5.3, is from `mod_blipcareBPmeterMud.json`. The second instead, named `from-ipv4-ihomepowerplug-2` and available in Listing 5.4, is from `mod_ihomepowerplugMud.json`.

[6] `https://github.com/emalinty/mud_files`, last visited on November 27th, 2023.

Listing 5.3: `from-ipv4-blipcarebpmeter-3`

```
{
    "name":"from-ipv4-blipcarebpmeter-3",
    "matches":{
        "ietf-mud:mud":{
            "local-networks":[
                null
            ]
        },
        "ipv4":{
            "protocol":17,
            "destination-ipv4-network":
                "255.255.255.255/32"
        },
        "udp":{
            "destination-port":{
                "operator":"gte",
                "port":50
            }
        },
        "eth":{
            "destination-mac-address":
                "ff:ff:ff:ff:ff:ff",
            "ethertype":2048
        }
    },
    "actions":{
        "forwarding":"reject"
    }
}

```

Listing 5.4: `from-ipv4-ihomepowerplug-2`

```
{
    "name":"from-ipv4-ihomepowerplug-2",
    "matches":{
        "ietf-mud:mud":{
            "local-networks":[
                null
            ]
```

```
8             },
9             "ipv4":{
10               "protocol":17,
11               "destination-ipv4-network":
12                   "255.255.255.255/32"
13            },
14            "udp":{
15               "destination-port":{
16                  "operator":"lte",
17                  "port":70
18               }
19            },
20            "eth":{
21               "destination-mac-address":
22                   "ff:ff:ff:ff:ff:ff",
23               "ethertype":2048
24            }
25         },
26         "actions":{
27            "forwarding":"accept"
28         }
29      }
30
```

In both ACEs, permission is granted to communicate within the local network using the local broadcast address (`255.255.255.255/32`), using Internet Protocol version 4 and using the UDP transport protocol. The first ace however rejects the traffic for ports greater or equal to 50, the second ace allows the traffic for ports lower or equal to 70. Therefore, the two rules are “partially matching”.

Since these rules are “partially matching”, it is possible to divide them in three parts (see Figure 5.3):

- Segment 1, from port 0 to port 49, isn’t subject to conflict resolution and remains "allow".
- Segment 2, from 50 to 70, is subject to conflict resolution depending on the option chosen by the user.
- Segment 3, from port 71 to port 65535, isn’t subject to conflict resolution and remains "reject".

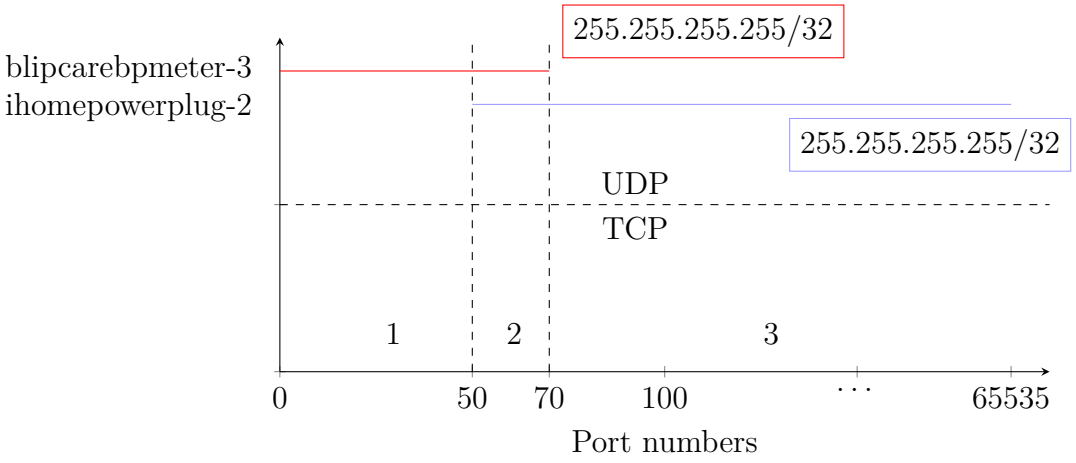

Figure 5.3: Conflicted rules

In this case, considering the Allow Takes Precedence (ATP) strategy, Segment 2 remains only in the permissive rule, creating Segment 1+2, and is eliminated from the prohibitive rule. The other parts remain intact, and here it is guaranteed that the behavior of the network will not change (see Figure 5.4).

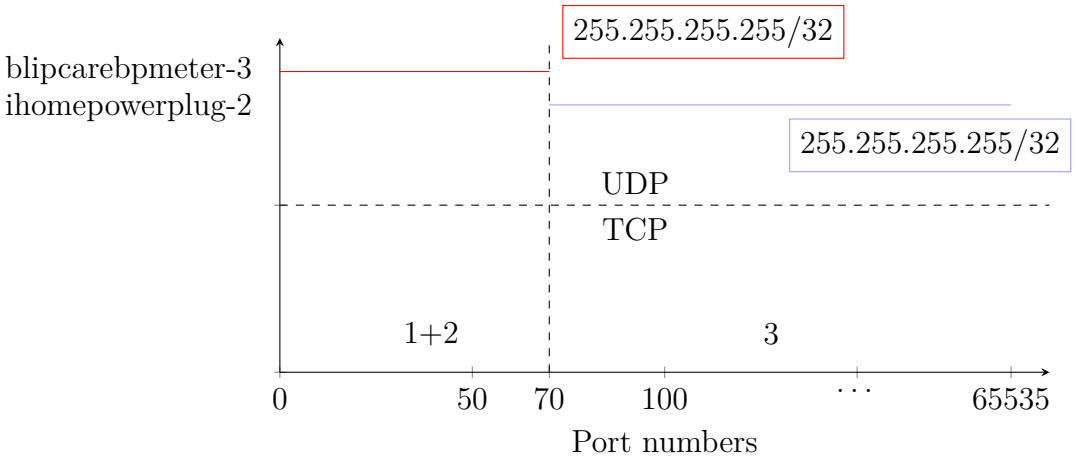

Figure 5.4: Conflict resolved

5.5 Discussion

The previous chapters have described the improvement and enhancement of the recent research study that led to the creation of the *MUD Aggregator* integration for Home Assistant. Thanks to this work, developers can independently specify

MUD policies for their plug-ins, adding an extra layer of protection to their Smart Home Gateway. The approach proposed in [7], giving the possibility of storing this consolidated gateway-level MUD file locally, can ensure MUD policy enforcement even if the manufacturer's servers are not available.

The approach proposed in this thesis aims to make more effective what is described above.

First, it is an effective support for developers regarding the creation of MUD snippets. These files can be generated with the help of specific tools or manually: in both cases, but especially in the second one, it is very useful and it produces very precise results, avoiding unexpected outcomes and unwanted exceptions regarding the JSON structure. Although the structure of a MUD rule is relatively simple, the MUD snippets are often very long and it is likely that an error could be found. Thanks to this work, the location and the reason of the error are signaled, making the correction operation quicker and simpler.

Second, this study aims to remove and resolve any type of ambiguity that may arise from merging different MUD snippets.

In terms of optimization, the reduction in the number of rules is a significant result: it allows policies to be enforced in less time, with all the benefits that this brings.

Instead, in terms of conflict resolution, the relevant result is that there are no more overlapping rules with different decisions: after the process described in Section 4.2.2, each packet will match only one rule. The relative limitation is that the result of the allowed address space may be altered.

As an example, supposing that the rule R_x is permissive, the rule R_y is prohibitive, and that the two rules are "exactly matching", if no operation of conflict resolution is performed, and R_x is before R_y, the packets matching these rules are allowed. If the conflicts are resolved and the user chooses Deny Takes Precedence (DTP), the packets are now dropped. The result is clearly different, and it is a direct consequence of the need to resolve conflicts between rules. In this work, it seemed more important to give priority to eliminating the ambiguities that can arise when merging the MUD snippets, so that the user can be aware that there are no overlapping rules in the final gateway-level MUD file.

Chapter 6

Conclusions

This thesis is based on a recent research proposal that improves IoT security in Smart Home Gateways through the introduction of Manufacturer Usage Description (MUD) standards into Smart Home plug-ins, allowing developers to specify the requirements of their plug-ins in a MUD-compliant way. The study focuses on validating the syntax of rules specified by developers for smart home plug-ins, resolving potential conflicts, and optimizing rules during the merging process into a unified gateway-level MUD file.

The proposed approach consists of an in-depth validation process in three stages: when the MUD draft is loaded, when a MUD snippet is found, and when the gateway-level MUD file is generated. The validation involves a JSON schema analysis based on RFC 8520, which ensures the correctness of the structure and fields of MUD files and snippets.

Rule optimization and conflict resolution are crucial aspects addressed in this research. The optimization process involves merging rules with identical actions, resulting in a significant reduction in the number of rules while maintaining the same allowed traffic space. Conflict resolution strategies such as Allow Takes Precedence (ATP), Deny Takes Precedence (DTP), Least Specific Takes Precedence (LSTP), and Most Specific Takes Precedence (MSTP) are used to handle conflicting rules with different actions, ensuring consistent and unambiguous rule enforcement.

Experimental results demonstrate the effectiveness of the proposed solution in all three aspects treated. The validation of MUD files outperforms an existing tool in terms of accuracy and comprehensiveness. The optimization process successfully reduces the number of rules, improving the speed of rule enforcement, while conflict resolution eliminates overlapping rules, providing consistency in policy

implementation.

In summary, this research contributes to improving the security of Smart Home Gateways by providing a validated, optimized, and conflict-free gateway-level MUD file, thus supporting the overall goal of establishing robust IoT security practices in Smart Home environments.

Acronyms

ACE Access Control Entry.

ACL Access Control List.

ATP Allow Takes Precedence.

C Correlated.

CD Completely Disjoint.

DHCP Dynamic Host Configuration Protocol.

DTP Deny Takes Precedence.

ECA Event Condition Action.

EM Exactly Matching.

FMR First Matching Rule.

HA Home Assistant.

HTTP Hypertext Transfer Protocol.

IETF Internet Engineering Task Force.

IM Inclusively Matching.

IoT Internet of Things.

IP Internet Protocol.

JSON JavaScript Object Notation.

LLDP Link Layer Discovery Protocol.

LMR Last Matching Rule.

LSTP Least Specific Takes Precedence.

MAC Media Access Control.

MSTP Most Specific Takes Precedence.

MUD Manufacturer Usage Description.

OS Operating System.

PM Partially Matching.

PBM Policy-Based Management.

PCAP Packet Capture.

PDP Policy Decision Point.

PEP Policy Enforcement Point.

RFC Request for Comments.

SHG Smart Home Gateway.

TCP Transmission Control Protocol.

TEAP Tunnel Extensible Authentication Protocol.

TLS Transport Layer Security.

UDP User Datagram Protocol.

URL Uniform Resource Locator.

YANG Yet Another Next Generation.

Bibliography

[1] Karen Rose, Scott Eldridge, and Lyman Chapin. «The internet of things: An overview». In: *The internet society (ISOC)* 80 (2015), pp. 1–50 (cit. on p. 1).

[2] Lionel Sujay Vailshery. *Internet of Things (IoT) total annual revenue worldwide from 2020 to 2030*. URL: https://www.statista.com/statistics/1194709/iot-revenue-worldwide/ (visited on Nov. 29, 2023) (cit. on p. 1).

[3] CR Srinivasan, B Rajesh, P Saikalyan, K Premsagar, and Eadala Sarath Yadav. «A review on the different types of Internet of Things (IoT)». In: *Journal of Advanced Research in Dynamical and Control Systems* 11.1 (2019), pp. 154–158 (cit. on p. 1).

[4] Eliot Lear, Ralph Droms, and Dan Romascanu. *Manufacturer Usage Description Specification*. RFC 8520. Mar. 2019. DOI: 10.17487/RFC8520. URL: https://www.rfc-editor.org/info/rfc8520 (cit. on pp. 2, 13, 25).

[5] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259. Dec. 2017. DOI: 10.17487/RFC8259. URL: https://www.rfc-editor.org/info/rfc8259 (cit. on pp. 2, 14).

[6] Paulus Schoutsen. *Home Assistant*. Sept. 2018. URL: https://www.home-assistant.io/ (visited on Nov. 29, 2023) (cit. on pp. 2, 9).

[7] Fulvio Corno and Luca Mannella. «A Gateway-based MUD Architecture to Enhance Smart Home Security». In: *2023 8th International Conference on Smart and Sustainable Technologies (SpliTech)*. 2023, pp. 1–6. DOI: 10.23919/SpliTech58164.2023.10193747 (cit. on pp. 2, 18, 60).

[8] Steven Waldbusser et al. *Terminology for Policy-Based Management*. RFC 3198. Nov. 2001. DOI: 10.17487/RFC3198. URL: https://www.rfc-editor.org/info/rfc3198 (cit. on p. 4).

[9] U Dayal, B Blaustein, A Buchmann, U Chakravathy, M Hsu, R Ledin, D McCarthy, A Rosenthal, and S Sarin. «HiPAC: a research project in active, time-constrained database management». In: *Interim Report* (1988) (cit. on p. 4).

[10] *Internet Protocol*. RFC 791. Sept. 1981. DOI: 10.17487/RFC0791. URL: https://www.rfc-editor.org/info/rfc791 (cit. on pp. 6, 30).

[11] Ehab S Al-Shaer and Hazem H Hamed. «Modeling and Management of Firewall Policies». In: *IEEE Transactions on Network and Service Management* 1.1 (2004), pp. 2–10. DOI: 10.1109/TNSM.2004.4623689 (cit. on pp. 7, 30).

[12] F. Valenza and M. Cheminod. «An optimized firewall anomaly resolution». In: *JOURNAL OF INTERNET SERVICES AND INFORMATION SECURITY* 10.1 (2020), pp. 22–37. DOI: 10.22667/JISIS.2020.02.29.022. URL: http://isyou.info/jisis/vol10/no1/jisis-2020-vol10-no1-02.pdf (cit. on p. 7).

[13] Habtamu Abie. «An Overview of Firewall Technologies». In: (Dec. 2000). URL: https://www.researchgate.net/publication/2371491_An_Overview_of_Firewall_Technologies (cit. on p. 8).

[14] C Muthu Ramya, Madasamy Shanmugaraj, and R Prabakaran. «Study on ZigBee technology». In: *2011 3rd international conference on electronics computer technology*. Vol. 6. IEEE. 2011, pp. 297–301 (cit. on p. 9).

[15] Behrang Fouladi and Sahand Ghanoun. «Security evaluation of the Z-Wave wireless protocol». In: *Black hat USA* 24 (2013), pp. 1–2 (cit. on p. 9).

[16] Shailandra Kaushik. «An overview of technical aspect for WiFi networks technology». In: *International Journal of Electronics and Computer Science Engineering (IJECSE, ISSN: 2277-1956)* 1.01 (2012), pp. 28–34 (cit. on p. 9).

[17] Chatschik Bisdikian. «An overview of the Bluetooth wireless technology». In: *IEEE Communications magazine* 39.12 (2001), pp. 86–94 (cit. on p. 9).

[18] Martin Björklund. *The YANG 1.1 Data Modeling Language*. RFC 7950. Aug. 2016. DOI: 10.17487/RFC7950. URL: https://www.rfc-editor.org/info/rfc7950 (cit. on p. 14).

[19] Tim Berners-Lee, Larry M Masinter, and Mark P. McCahill. *Uniform Resource Locators (URL)*. RFC 1738. Dec. 1994. DOI: 10.17487/RFC1738. URL: https://www.rfc-editor.org/info/rfc1738 (cit. on p. 14).

[20] Eric Rescorla. *HTTP Over TLS*. RFC 2818. May 2000. DOI: 10.17487/RFC2818. URL: https://www.rfc-editor.org/info/rfc2818 (cit. on p. 17).

[21] Ralph Droms. *Dynamic Host Configuration Protocol*. RFC 2131. Mar. 1997. DOI: 10.17487/RFC2131. URL: https://www.rfc-editor.org/info/rfc2131 (cit. on p. 17).

[22] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. May 2008. DOI: 10.17487/RFC5280. URL: https://www.rfc-editor.org/info/rfc5280 (cit. on p. 17).

[23] Hao Zhou, Nancy Cam-Winget, Joseph A. Salowey, and Steve Hanna. *Tunnel Extensible Authentication Protocol (TEAP) Version 1*. RFC 7170. May 2014. DOI: 10.17487/RFC7170. URL: https://www.rfc-editor.org/info/rfc7170 (cit. on p. 17).

[24] «IEEE Standard for Local and metropolitan area networks – Station and Media Access Control Connectivity Discovery». In: *IEEE Std 802.1AB-2005* (2005), pp. 1–176. DOI: 10.1109/IEEESTD.2005.96285 (cit. on p. 17).

[25] Kevin Yeich and Daniel Weller. *Open Source Manufacture Usage Description*. 2018. URL: https://osmud.org (visited on Nov. 29, 2023) (cit. on pp. 17, 18).

[26] Pablo Neira Ayuso, Jozsef Kadlecsik, Eric Leblond, Florian Westphal, Arturo Borrero González, and Phil Sutter. *The netfilter.org "iptables" project*. 1998. URL: https://www.netfilter.org/projects/iptables/index.html (visited on Nov. 29, 2023) (cit. on p. 17).

[27] *The linux kernel*. URL: https://docs.kernel.org/security/self-protection.html (cit. on p. 17).

[28] José L. Hernández-Ramos, Sara N. Matheu, Angelo Feraudo, Gianmarco Baldini, Jorge Bernal Bernabe, Poonam Yadav, Antonio Skarmeta, and Paolo Bellavista. «Defining the Behavior of IoT Devices Through the MUD Standard: Review, Challenges, and Research Directions». In: *IEEE Access* 9 (2021), pp. 126265–126285. DOI: 10.1109/ACCESS.2021.3111477 (cit. on p. 18).

[29] OpenWRT. *Documentation*. 2023. URL: https://openwrt.org/ (cit. on p. 18).

[30] Eliot Lear and Vafa Andalibi. *MUD Maker*. URL: https://mudmaker.org (visited on Nov. 29, 2023) (cit. on p. 21).

[31] Ajay Chandra V Gummalla and John O Limb. «Wireless medium access control protocols». In: *IEEE Communications Surveys & Tutorials* 3.2 (2000), pp. 2–15 (cit. on p. 26).

[32] Herman Slatman. *mud.yang.go*. Oct. 2016. URL: https://github.com/hslatman/mud.yang.go (visited on Nov. 29, 2023) (cit. on p. 27).

[33] Guido Van Rossum et al. «Python Programming Language.» In: *USENIX annual technical conference*. Vol. 41. 1. Santa Clara, CA. 2007, pp. 1–36 (cit. on p. 27).

[34] Alan AA Donovan and Brian W Kernighan. *The Go programming language*. Addison-Wesley Professional, 2015 (cit. on p. 27).

[35] Eliot Lear, Ralph Droms, and Dan Romascanu. *Manufacturer Usage Description Specification*. Internet-Draft draft-ietf-opsawg-mud-21. Work in Progress. Internet Engineering Task Force, May 2018. 60 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-opsawg-mud/21/ (cit. on p. 27).

[36] Florence Benoy and Peter Rodgers. «Evaluating the comprehension of Euler diagrams». In: *2007 11th International Conference Information Visualization (IV'07)*. IEEE. 2007, pp. 771–780 (cit. on p. 28).

[37] Giorgos Papastergiou et al. «De-Ossifying the Internet Transport Layer: A Survey and Future Perspectives». In: *IEEE Communications Surveys & Tutorials* 19.1 (2017), pp. 619–639. DOI: 10.1109/COMST.2016.2626780 (cit. on p. 30).

[38] *Transmission Control Protocol*. RFC 793. Sept. 1981. DOI: 10.17487/RFC0793. URL: https://www.rfc-editor.org/info/rfc793 (cit. on p. 30).

[39] *User Datagram Protocol*. RFC 768. Aug. 1980. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/info/rfc768 (cit. on p. 30).

[40] Vafa Andalibi, Jayati Dev, DongInn Kim, Eliot Lear, and L. Jean Camp. «Is Visualization Enough? Evaluating the Efficacy of MUD-Visualizer in Enabling Ease of Deployment for Manufacturer Usage Description (MUD)». In: *Annual Computer Security Applications Conference*. ACSAC '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 337–348. ISBN: 9781450385794. DOI: 10.1145/3485832.3485879. URL: https://doi.org/10.1145/3485832.3485879 (cit. on p. 44).

[41] Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili, Matthew Roughan, and Vijay Sivaraman. «Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles». In: *Proceedings of the 2018 Workshop on IoT Security and Privacy*. IoT S&P '18. Budapest, Hungary: Association for Computing Machinery, 2018, pp. 8–14. ISBN: 9781450359054. DOI: 10.1145/3229565.3229566. URL: https://doi.org/10.1145/3229565.3229566 (cit. on p. 44).

[42] Guy Harris and Michael Richardson. *PCAP Capture File Format*. Internet-Draft draft-ietf-opsawg-pcap-03. Work in Progress. Internet Engineering Task Force, July 2023. 10 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-opsawg-pcap/03/ (cit. on p. 44).

[43] *Withings*. URL: https://www.withings.com/ (visited on Nov. 29, 2023) (cit. on p. 52).

[44] *Netatmo*. URL: `https://www.netatmo.com/` (visited on Nov. 29, 2023) (cit. on p. 52).

[45] *TP-Link*. URL: `https://www.tp-link.com/` (visited on Nov. 29, 2023) (cit. on p. 52).

[46] *Belkin*. URL: `https://www.belkin.com/` (visited on Nov. 29, 2023) (cit. on p. 52).

[47] *Awair*. URL: `https://www.getawair.com/` (visited on Nov. 29, 2023) (cit. on p. 52).

[48] *Pix-Star*. URL: `https://www.pix-star.com/` (visited on Nov. 29, 2023) (cit. on p. 52).

Appendix A

Setup of MUD Aggregator

This appendix explains how to install and correctly set up in Home Assistant the extended version of the *MUD Aggregator* integration presented in this thesis.

The integration can be found and downloaded at this link: `https://github.com/LucaMannella/HomeAssistant-MUD-Aggregator`.

The user must create a folder named `custom_components` under the folder `config` if it doesn't already exist. As the name suggests, `custom_components` contains all non-default integrations (i.e., all the integrations not maintained by Home Assistant core team). Here the user has to insert the downloaded folder of *MUD Aggregator*.

Now the installation is almost finished. The last step to be completed is to set the configuration parameters for specifying the validation strategy for the MUD snippets and for the solving the conflicts.

To do this, the user must edit the `configuration.yaml` file in the `config` folder (see A.1).

Listing A.1: Configuration code for integrating the *MUD Aggregator* in the `configuration.yaml` file

```
1    sensor:
2      - platform: mud_aggregator
3        validation: 1
4        policy_option: 3
```

Regarding validation, the user can choose between two options in case of an error:

- **0**, the badly-written MUD snippet is skipped and the relative rules are not inserted in the gateway-level MUD file.
- **1**, the entire process is stopped.

In both cases, Home Assistant will display a notification indicating the problematic file and the type of error.

Instead, regarding the conflict resolution process, the user can choose between four possible options (see Section 2.1):

- **1**, Allow Takes Precedence (ATP).
- **2**, Deny Takes Precedence (DTP).
- **3**, Most Specific Takes Precedence (MSTP).
- **4**, Least Specific Takes Precedence (LSTP).

The constructor method of this sensor is the `__init__` method, which creates the gateway-level MUD file at the launch of the integration.

Listing A.2: The `__init__` method

```
1     def __init__(self, params):
2         self._mud_gen = MUDAggregator()
3         self._mud_gen.generate_mud_file()
4         self._mud_gen.expose_mud_file(self._interface)
```

Since the integration *MUD Aggregator* is a sensor, Home Assistant periodically generates the gateway-level MUD file thanks to the `update` method, called by default every 30 seconds (see Listing A.3).

Listing A.3: The `update` method

```
1     def update(self) -> None:
2         integration_list = self.hass.data["integrations"]
3         self._mud_gen.generate_mud_file(integration_list)
4         self._mud_gen.expose_mud_file(self._interface)
```

In order to debug the integration and see the creation of the gateway-level MUD file when the user wants, it is possible to configure the integration as a button. When the button is pressed, the whole process starts. This is how the experiments were done in this thesis.