Fine-grained Access Control for Internet of Things Smart Spaces Driven by User Inputs

by

Mohammed Al-Shaboti

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2021

Abstract

The increasing use of Internet of Things (IoT) devices raises security and privacy concerns. In smart spaces, multiple IoT devices are simultaneously used to fulfil user activity functions. However, these devices exhibit several security vulnerabilities that can compromise smart space security and privacy. The ability of fine-grained control network access in IoT devices and application messages can significantly reduce the risk resulting from the exploitation of IoT vulnerabilities due to unauthorised access, thereby improving smart space security. A well-recognised approach in the literature for IoT access control is to use pre-defined access policies to allow the necessary connections for a device to function correctly. However, these policies allow access to all device functions (i.e. coarse-grained access) including those functions that are not used by any user activity.

The overall goal of this thesis is to develop an access control framework and techniques to achieve fine-grained access policies by using user inputs. The user inputs will be utilised to select devices to fulfil user activities aiming to build an access policy from the minimum access required for each device function. In this thesis, the use of user inputs to meet user security and privacy requirements in single- and multi-user smart spaces is studied.

The main contributions are as follows: first, an access control framework that enables users to tailor IoT device policies to meet their security and privacy requirements is proposed. Validation results of the framework show the effectiveness of integrating user access rules into the existing security countermeasures (i.e. pre-defined policies and intrusion detection systems – IDS) to enforce user security and privacy.

Second, the problem of selecting preferable devices to fulfil user activity functions is formulated as an optimisation problem. The optimisation problem is then solved by local and global optimisation searching algorithms that are guided by a developed user preference quantified model. The results show that global optimisation search algorithms such as Genetic Algorithm (GA) find the solution more effectively and efficiently than local search algorithms such as simulated annealing and hill-climbing.

Third, sharing access control for multi-user smart spaces is proposed. Traditional access control that considers a single user is not suitable for multi-user smart spaces, where users share their IoT devices. The sharing between multiple users poses challenges different than in single-user access control. For example, users may abuse using shared devices and use vulnerable ones. This thesis addresses these two challenges through two contributions. First, it proposes a novel sharing policy language that enables users to precisely define their sharing policy. Second, this thesis formulates the sharing policies as constraints in the context of an optimisation problem with the objective function that maximises the use of secure devices. Results show that the IoT sharing issue can naturally be translated into an integer linear programming (ILP) problem and effectively solved using off-the-shelf ILP solvers.

Fourth, this thesis explores the feasibility and practicality of the fine-grained access policy enforcement through a smart home case study. A case study is built using a hub-based architecture that uses Web of Things (WoT) technology. WoT provides a device semantic description that includes device functions with the corresponding Uniform Resource Identifier (URI) which is used to build access control policies. The case study results show that policy enforcement can be effectively achieved by directing network traffic through a device proxy for each IoT device to enforce application access control without introducing statistically significant overhead on the user activity running time.

In summary, this thesis studies the use of user inputs to derive fine-grained access control in smart spaces. For a single-user access control system, this thesis considers using manual rules and user preferences in small and dense smart spaces, respectively. For a multi-user access control system, this thesis proposes a secure sharing system supported by a sharing policy language to share and use IoT devices securely. For each scenario analysed, user input is utilised to derive fine-grained access policies. Enforcement of these policies has been explored by implementing a smart space case study using WoT technology. The overall results show that user preferences and sharing policies can be used to derive fine-grained access policies that are transparent to users and meet their security and privacy requirements.

Dedication

To my parents, sisters and brothers, and my beloved wife and daughter.

Acknowledgements

Praise be to Almighty ALLAH alone, Who created us empty of any knowledge bestowed upon us hearing, vision, and intellect so that we must be thankful to Him. Praise be to Him alone, Who let wonderful, encouraging and supporting people come into life without whom this journey would have been impossible.

First and foremost, I would like to express my deep sense of gratitude to my supervisors **Dr Ian Welch** and **Dr Aaron Chen**. I am indebted to them for their outstanding supervision, support, encourage, and mentoring. I feel lucky to have had the opportunity to work with Ian and Aaron, who put my feet on the door of research. I extend my gratitude to all my teachers from primary school up to my current supervisors.

I am deeply grateful to my family, in particular, my parents (Mahmood and Raqiba), who always strive to help me become better and better. My thanks to my beloved wife Nawal and daughter Khadijah, and all my brothers and sisters for supporting me during my PhD journey.

I cannot be thankful enough to Almighty, Who gave me friends like family. My friends, my brothers Baligh, Mohamed Arif, Aditya, Harith, Masood, Harisu, Mahdi, Junaid, Abbasi, and all other friends and colleagues at Victoria University of Wellington.

This acknowledgement would be incomplete without thanking all staff members of the School of Engineering and Computer Science.

List of Publications

1. Al-Shaboti, M., Chen, A., & Welch, I. Achieving Optimal and Secure Sharing of IoT Devices in Multi-User Smart Spaces. LCN 2020 45th IEEE Conference on Local Computer Networks. (Accepted)

2. Al-Shaboti, M., Welch, I., & Chen, A. (2019, July). IoT Application-Centric Access Control (ACAC). In Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (pp. 685-687).

3. Al-Shaboti, M., Chen, A., & Welch, I. (2019, August). Automatic Device Selection and Access Policy Generation based on User Preference for IoT Activity Workflow. In 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE) (pp. 769-774). IEEE.

4. Al-Shaboti, M., Welch, I., Chen, A., & Mahmood, M. A. (2018, May). Towards secure smart home iot: Manufacturer and user network access control framework. In 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA) (pp. 892-899). IEEE.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

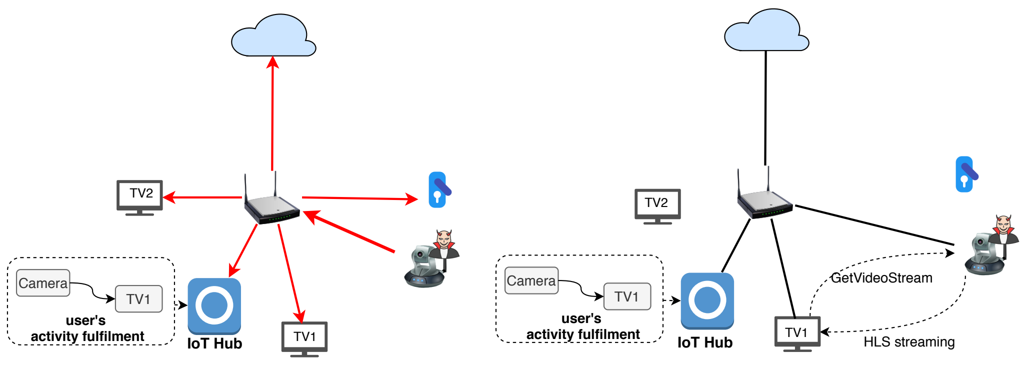
The Internet of Things (IoT) paradigm is a technological revolution in the era of computing [112, 53]. The Institute of Electrical and Electronics Engineers (IEEE) [118] defines the IoT as a network that connects things to the Internet, wherein the thing has a unique identity, sensing/actuating capabilities, and potentially programmability. In the IoT context, things are seamlessly connected and provide contextual services [85]. The number of deployed IoT devices is growing fast. Gartner forecasts the number of IoT endpoints to reach 5.8 billion in 2020, a 21% increase compared to 2019 [59]. It also forecasts that consumer devices will account for the highest number of IoT endpoints.

IoT devices are the building blocks of smart spaces. Hundreds of IoT devices are expected to be located in a single residential house in the near future [60]. A smart space is formed by utilising IoT device sensing, communication, and computation capabilities to provide new services [109]. *Smart space* in this thesis refers to an environment where users and IoT devices coexist, such as smart homes, offices, and buildings. Although many proposed ideas can be applied in Industrial IoT (IIoT), this thesis mainly focuses on consumer smart space to demonstrate the proposed methods.

One of the key features of a smart space is the ability to use multiple devices to support user activity [6]. Individual IoT devices have limited functions and provide simple services. Users want to build services that enhance their smart space experience [189, 194]. For example, as illustrated in Figure 1.1, instead of accessing the IP camera from their mobile, users can build an activity where the camera automatically streams the video content to a display device of their choice. We will refer to such service as *user activity*. Implementing such activities can be realised by using existing IoT frameworks.

Figure 1.1 shows a common smart space architecture that is used to implement user activities. This architecture includes three main components: IoT devices, a hub, and IoT clouds, saves the IoT status at the cloud. IoT devices are the physical devices that reside within the smart space and have sensing and actuating capabilities. They connect to their IoT cloud services to synchronise their status and receive control commands. The IoT hub is a centralised controller that integrates all IoT devices. Users add their IoT devices to the hub and authorise it to control them, often using OAuth protocol [76]. Thereafter, they can use the hub to build activities that make their life easier, more distinctive, and pleasant. For example, streaming a video from a camera to a smart TV, as shown in 1.1. Examples of hubs include Mozilla WebThings Gateway [124], Amazon echo [10], and SmartThings [84].

In order for IoT devices to execute user activities they require network access. An IoT device network access can be (a) coarse-grained in which the device may have network access more than what is necessary to do its function; (b) fine-grained in which the device has only the necessary network access to do its function. For example, Figure 1.1 shows a user activity that stream IP camera video to the TV1. However, in Figure 1.1 (a) the IP camera has access not only to the TV1 but also to other devices that are not required to fulfil its function in the user activity, this is called coarse-grained access. In contrast, in Figure 1.1 (b) the IP camera has access only to the TV1 and the Hub, which is required to fulfil its function in the user activity, this is called fine-grained access.



(a) A malicious IP camera with network based access to the Internet and to other devices in the smart space.

(b) A malicious IP camera with an application access to the Internet and to other devices in the smart space.

Figure 1.1: Smart space in (a) coarse-grained and (b) fine-grained access control.

Although IoT devices enhance the user experience, they exhibit many security vulnerabilities. The top ten most important IoT vulnerabilities that are listed in the Web Application Security Project (OWASP) [139] includes weak passwords and insecure network services vulnerabilities on legacy devices. A security report [46] indicated that 70% of IoT devices used unencrypted network protocols, and 60% of IoT user interfaces were vulnerable. Security researchers [86, 105] assessed several IoT devices and concluded that all devices have some type of vulnerability.

This research focuses on two issues in the current smart space architecture [152, 187]. The first relates to coarse-grained access control, where a device has access based on what it can be used for, regardless of user activity. For example, the camera depicted in Figure 1.1 (a) can stream its video content to a display device (TV1, TV2), and it can also open the door. However, the user activity only uses the camera for streaming video to TV1 only; hence, the camera has network access more than what is required by user activity (i.e. it has coarse-grained access). In contrast, by applying fine-grained access control, the camera has access to the devices (i.e. TV1 and the Hub) that are required for video streaming, as in Figure 1.1 (b). Moreover, the issue of coarse-grained access is even more severe in a shared smart space, as devices not only have coarse-grained network access but also are shared with other users with full access.

The second issue is that users have to explicitly specify the devices to fulfil user activities. It has been forecasted that smart spaces, such as smart homes, will be occupied by hundreds of IoT devices in the future [63, 60]. This thesis will refer to a smart space that includes hundreds of IoT devices as a dense smart space. In a dense smart space [180, 56], users have to search through a large number of IoT devices to manually specify the devices to fulfil their activities. Moreover, when activities are built using a specific set of devices, they are not transferable, which implies that they cannot be used in another environment. For example, for the activity of preparing coffee, a user has to build it for home using a specific coffee machine and build it again for other spaces using their respective coffee machines. If user activities were transferable, users would be able to run their activities in different places, and they would be automatically fulfilled using existing devices that support their required functions.

The coarse-grained access control in smart spaces may allow the exploitation of IoT vulnerabilities [181, 200]. Therefore, recent studies on IoT security focus on mitigating the security risk of compromising IoT devices through access control [78]. Coarse-grained access control often works in the network layer, using the IP address, IP transport protocol, and port to define the legitimate local and remote connections for a device. For example, in Figure 1.1, the malicious IP camera can connect to its cloud server using a specific protocol, and also to all other devices in the local network. This is an example of coarse-grained control as the camera has access to local devices (i.e. TV2, smart lock) that are not currently linked with to them in any user activities.

Current IoT platforms often support coarse-grained access control, wherein applications are granted access to more resources than the minimum required to perform their functions [181, 48, 91]. In [48], the authors show that the SmartThing platform grants the SmartApp full access to devices, although its function requires limited access. Zhou et al. [200] analysed a commonly used smart space platform and found several design flaws that allow device hijacking and Denali of Service (DoS) attacks. IETF proposed Manufacturer Usage Description (MUD) to define a network-wide access policy for the legitimate connections possible for an IoT device [101]. For example, in Figure 1.1, the MUD policy can define the external network access for the IP camera for external connections to a cloud domain using a specific transport protocol and port. However, it cannot define the local devices to which the IP camera can connect. Therefore, if compromised, the camera can spy on the network activities and/or infect other devices.

In order to securely fulfil user activity, fine-grained access control is required. The goal is to apply the least privilege access by granting devices the minimum access permission necessary to fulfil user activities. Figure 1.1 illustrates the fine-grained access control, wherein devices are only granted access if they are currently used in user activities. For example, the IP camera does not have network access to the door lock, as no user activity requires to use the door lock. However, it has network access to the smart TV. Moreover, as the user uses the IP camera to stream videos to the TV, the IP camera access to the TV is limited to the streaming function.

Research indicates that the main concerns of users are security and privacy as they share the same space with the IoT devices [75, 198, 197]. A study reports that 90% of smart home devices collect personal information [46]. From the Internet community perspective, adversaries may utilise vulnerable IoT devices to form a botnet and then use it to launch malicious activities on the Internet (e.g. Distributed denial of service DDoS attack). Another study [111] showed that only eight IoT devices were able to amplify an attack to inflict 1.2 Mbps of data on the victim in a DOS scenario. Hundreds of thousands of compromised IoT devices can inflict hundreds of Gbps; for example, the Mirai botnet, which infected 600000 IoT devices [33], exceeded 660 Gbps in volume [96].

IoT vulnerabilities will continue to be an issue in the foreseen future; they are inherited from the complex IoT ecosystem combined with its supply chain. The ecosystem includes the perception, network, and application layers, each of which has its own vulnerabilities [88]. IoT manufacturers use commodity hardware and third-party software, not produced by them. Hence, vulnerabilities in these components could allow attackers to compromise millions of devices from different vendors [115].

Fine-grained access control is an effective countermeasure to mitigate risks arising from IoT vulnerabilities. IoT devices have been frequently exploited by attackers, who compromise user data and threaten the Internet infrastructure. Fine-grained access control is necessary to limit the damage that can be inflicted if IoT devices are compromised. The lack of this type of control is one of the main reasons that enables adversaries to exploit IoT vulnerabilities [41, 1, 133, 136, 94, 135]. An adversary can compromise user privacy [154, 175], use the IoT devices as an entry point to compromise the entire network [169, 36], cause a physical breach [196, 193], or compromise user privacy [154]. By enforcing fine-grained access control, IoT devices will have minimum access to other devices that is necessary to support user activities. Hence, when compromised, their impact on the network will be minimised as well.

To support fine-grained access control and automatic user activity fulfilment requirements, we have considered four user inputs: manual network access rules, user activity as a functional workflow, user preference, and user sharing policy. These user inputs are manual network access rules, user activity, user preference, and sharing policy, as shown in Figure 1.2. The network access rules input provides a flexible option to achieve fine-grained access control in a small smart space. However, manual rules do not scale and can be a burden to users in larger and more complex smart spaces. In this case, it is necessary to support fine-grained access control and user activity automation requirements in a transparent manner, [50] without explicit user input. Therefore, devices will be automat-

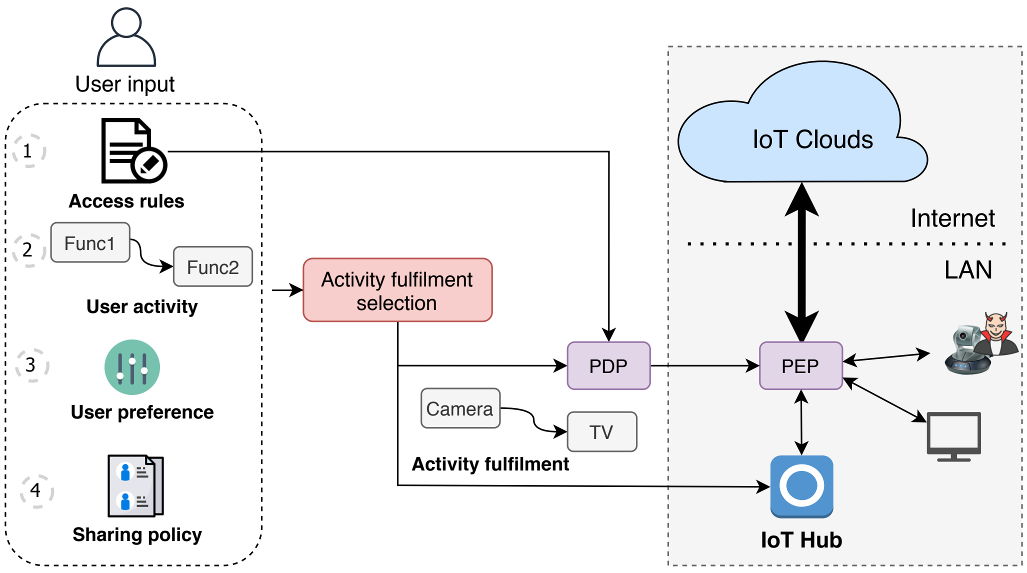


Figure 1.2: Smart space system architecture with fine-grained access control based on user inputs. The Policy Decision Point (PDP) makes the authorisation access decision and the Policy Enforcement Point (PEP) enforces it.

ically selected based on their ability to fulfil a user activity, which is represented as a workflow of user activity function requirements. The functional representation of user activity can be represented as Unified Modelling Language (UML) or Entity Relationship (ER) Diagram. However, the workflow representation is more natural for describing the problem and the way that user activities are represented in existing frameworks like NodeRed [142].

The user activity will be fulfilled considering user preferences and sharing policies. Figure 1.2 presents a functional representation of user activity (no. 2) and how it is used along with user preferences and sharing policy to guide the selection of the camera and TV devices. Subsequently, a fine-grained network and application access policy will be generated to support the selected devices to execute the user activity.

1.1 Motivation

There are three main driving questions for this research: (1) Why is it essential to consider user input? (2) What type of user inputs should be used to enable automatic user activity fulfilment and derive fine-grained access control? (3) How to use user input to automatically fulfil user activities and derive fine-grained access control?

User input must be considered when developing a fine-grained access system for smart spaces [44, 51, 201, 38]. As stated by [44], there are several factors that limit the usefulness of an automated security system that does not involve the user in the security decision. In smart homes, these reasons are summarised as follows:

1. Without user input, automated systems cannot make a security decision based on social relationships such as family, friendship, and guest relations. Instead, a set of pre-defined access policies (e.g. roles) is generated for each group of users, which cannot cover all social relationships. Such roles also treat every member in the group equally without considering the trust between users. For example, based on the trust between users (socially dependent), some users may allow specific guests to access their IoT devices [80, 58].

2. Automated access control systems that do not consider user input can disrupt user activities and result in a poor user experience. Defining an IoT access policy without considering its function in user activity may result in blocking vital connections, thus; disturbing user activities. For example, anomaly-based IDS is prone to false positive, which means it may detect benign connections as malicious, which results in dropping a connection that disturbs user activities [9, 17]. In contrast, to mitigate disturbances in user activities, an automated system may apply coarse-grained policies that allow unnecessary communications. For example, deriving an access control

policy based on the analysis of IoT metadata or network traffic may not cover all legitimate device behaviours [181, 73].

3. The owners of the smart space network are often the users themselves; hence, they are usually motivated to be involved in securing their network.

Most of the existing research mainly focuses on defining policies for IoT devices without involving user inputs. This includes MUD and similar policies that are derived based on benign traffic of IoT devices [73, 17, 172]. Other works focus on anomaly detection or outsourcing access control to a third party. Access control is often enforced in the network layer, as it can be applied to heterogeneous IoT devices. Network access control enforcement can be categorised into perimeter-based and network-wide access control. The perimeter-based access control is enforced at the boundary of the smart space network [192, 169], at the edge of the local network using Network Address Translation (NAT) firewall or at the ISP edge switch [169]. Perimeter access control does not provide control over internal threats. In contrast, the network-wide access control is enforced within the devices (i.e. inside the network) [15, 77]. Hence, it provides control over the connections between IoT devices as well as at the perimeter of the network. Using user input to derive a fine-grained access policy has not been thoroughly investigated. Fine-grained access control can be derived from user activities such that devices can be granted access based on their roles in fulfilling user activities. The question is: what type of user inputs are required to derive fine-grain access control for IoT devices?

Manual rules as user input provide advanced users with an explicit and flexible means of specifying their security requirements for a small smart space. Using access rules, users can directly restrict access to devices that they want to protect (e.g. for privacy reasons) such as an IP camera. Although it is not suitable for novice users, manual rules input is the most accurate method of obtaining user input. The connections between devices depend on the user activities [40]. Hence, in a simple environment, advanced users [75] can customise the pre-defined policies [172, 73], such as MUD [101], to meet their security and privacy requirements. Smart spaces also include other security services such as IDS [161, 134, 40, 116]. Thus, there are three main security components for smart spaces: user input, pre-defined IoT policies, and network security middleboxes, which need to be incorporated when designing smart space security system.

In a dense smart space, user preferences can be used to provide automatic activity fulfilment and transparent access control. It is challenging for novice and even advanced users to manually maintain fine-grained access control for such a large number of IoT devices. In this scenario, user preference can be used to automatically select the devices that fulfil user activities [113]. User preference can captures which devices a user prefers to use for different functions. Moreover, users trust IoT device manufacturers [199] to protect their privacy; hence, their preference capture their privacy and security decisions. Consequently, access policies can be generated for each user activity transparently, without explicit input from users, at the expense of requiring a user preference.

User preferences can be used to automatically derive a fine-grained access policy and fulfil user activity. Therefore, user activity can be defined as a functional workflow that can be fulfilled using the devices that maximise user preferences. Automatic device selection to fulfil user activity can be formulated as an optimisation problem to maximise user preferences on the selected devices. Thus, given the selected devices to fulfil user activity, a fine-grained access policy can be automatically generated for these devices.

In a shared smart space, users' sharing policies can be used to express users' sharing decisions. The user preference input does not capture the relationship among users and what devices a user wants to share with others. In a multi-user shared smart space, users share devices based on the trust among them [58, 198]. In addition, they share their IoT devices, such as smart TVs and smart speakers, for altruistic or economic reasons. Although a single user can link to and use multiple devices, a single device cannot be linked to multiple user accounts (the IoT cloud revokes an existing binding when it receives a new binding request) [29, 200]. In existing IoT frameworks, IoT devices are either shared with everyone (e.g. using the primary account) [171] or are only used by a single user [26, 90].

Sharing devices with full access violates the principle of least privilege and is a high-security risk [26]. Instead, it is more sensible to share under certain restrictions to limit the risk of abuse [198, 160]. However, existing IoT architectures are designed for a single user and do not support secure sharing [90, 165]. In particular, there are two issues we want to study: from the perspective of the owners, devices can only be shared with coarse-grained access, i.e. they cannot share a device for a particular function. Secondly, from the a user' perspective who is using shared devices, there is a risk of using a shared device that is vulnerable. For example, if there are two shared IP cameras, one with out dated vulnerable firmware, while the other with up to date firmware, a user's activity may utilise any of them rather than preferring the latter one.

User sharing language is required to enable users to express their sharing decisions on the device function level. In shared smart spaces, users share their devices under trust-based policies [58], which are not guaranteed to be followed. Policy languages are often used to express user policies, such as expectation [193] and safety policies [104] in smart spaces. Given user activities, with a dataset of the devices' functions and owners, an optimisation problem can be formulated to fulfil user activities using the most secure devices subject to users' sharing policies.

1.2 Problem Statement

The main problem addressed by this thesis is how to develop a framework and techniques to use user inputs to achieve automatic user activity fulfilment and fine-grained access control in smart spaces. This problem can be divided into three main scenarios based on the complexity and the number of users of the smart space:

1. The first scenario is a small smart space where a single user uses a few devices to fulfil infrequently changed user activities. In this scenario, users will require a way to specify what devices should connect to what. The assumption here is that users are familiar with access control rules that are usually use in a home router firewall [182]. For example, a user may want to limit access to an IP camera only to his/her own device. Specifying the rules that overwrite existing IoT device policies and present them to a user in a simple way is the problem in this scenario.

2. The second scenario is a dynamic and dense smart space where a single user uses hundreds of devices to fulfil frequently changed user activities. The smart space is growing in size (i.e. number of devices) and becoming more complex in terms of IoT device capabilities. This poses a challenge for users when automating their activities. Users are not always interested in manually specifying the underlying devices to fulfil their activities. Nevertheless, they have to search through a large number of IoT devices to identify the suitable devices that can fulfil their activity and meet their preferences. The challenge is how to select the devices that fulfil user activity functional requirements while considering user preferences.

3. The third scenario is a shared smart space where multiple users share their devices under certain conditions. In this scenario, a sharing policy language is required to capture users' sharing policies. There are two challenges in this scenario: how to develop a sharing policy language that captures user sharing conditions and use it to derive fine-grained access control; and how to mitigate the risk of using vulnerable shared devices when fulfilling user activities. For example, a

user wants to share the thermostat on/off function, but not the temperature setting function. Alternatively, a user may want to make a video conference call using the most secure camera device among the available shared devices.

1.3 Research Questions

The present research will address the following research questions:

1. *How can users define access control rules to customise pre-defined coarse-grained policies in smart spaces?*

   As access control depends on how user activities use devices, the pre-defined policies cannot be fine-grained. For example, a MUD for an IP camera may allow its access to its remote manufacturer cloud server and all local devices that belong to the same manufacturer. The local access can be customised by a user only to the devices that he/she is actually using with the IP camera. Users can provide input to customise these policies to meet their security and privacy requirements. However, using user input to customise pre-defined IoT policies is challenging in terms of how to allow users to specify the rules and how to represent the rules. More importantly, users may mistakenly add rules that disrupt user activities (i.e. block necessary connections) or add too coarse-grained rules.

   The following sub-questions help address this research question:

   (a) How can user access rules be specified and be presented in a simple manner??

   Advanced users are familiar with the user interface of their NAT firewall, which they use to allow or deny traffic passing through their router. Therefore, a similar user interface will be used to obtain user rules for local access between devices (i.e. network-wide access rules). Usability is not the main focus of this research, but common user interface techniques will be used for any user interface components. For example, to present existing rules in the network; a table that lists all rules similar to the router firewall and a visualised view that shows the network topology with the rules on each link.

(b) How can user-specified rules be used to customise pre-defined device policies?

The pre-defined policies can provide the base policy for access control that users can overwrite. However, as users may mistakenly/unknowingly input rules that may disrupt user activities, the pre-defined policies serve as the base access control backup that can be used to reset the access control to the correct state.

2. *How to automatically fulfil user activity and generate a fine-grained access policy in a dense and complex smart space using user preferences?*

Achieving fine-grained access control is a challenging task in a dense and complex smart space. A dense smart space includes hundreds of devices and offers multiple alternatives to support user activity. Therefore, user preference can be used as user input to determine which candidate device should be used to fulfil user activity and subsequently generate fine-grained access control.

The goal is to automatically select devices that fulfil user activities while considering user preferences. To automatically select devices, a new user activity representation is required to decouple the user activity functional requirements from the underlying devices. This representation will enable automatic searching for the devices that fulfil the user activity while optimising user preference.

To solve this task, we need to answer the following sub-questions:

(a) How can user activity be presented to enable flexible device fulfilment?

The existing user activity representations [124, 52] does not allow flexible device fulfilment. Users define their activities by specifying the devices that should be used to execute them. This representation hinders the dynamic changing of devices to meet user preferences without rebuilding the user activity. Therefore, a new user activity representation is required to decouple the user activity functional requirements from the underlying devices that can be used to fulfil them.

(b) How to select the devices that fulfil user activities while considering user preference? And how can user activity be used to generate fine-grained access control?

The problem of searching for a set of devices to fulfil user activity functions while satisfying user preference is formed as an optimisation problem. The user activity will be represented by a functional workflow, which can be fulfilled by many candidate devices. The selection criteria are based on a user preference model that quantifies which devices the user prefers and for which functions. Therefore, search algorithms will be used to find the set of devices that fulfil user activities and maximise user preferences.

The selected devices need to be supported by a fine-grained access policy. The policy allows each device to make the minimum connections necessary to fulfil the user activity. For the policy to be generated, we assume that each device function has an associated set of network requirements. Hence, the fine-grained access policy is built based on the network requirements of each device function of the selected devices.

3. *How to design a system that enables users to share their devices under cer-*

tain conditions and use the most secure available devices to fulfil their activities?

This question addresses the scenario in which IoT devices are shared between multiple users. In particular, it studies two aspects of this scenario. The first aspect is how devices can be shared for a specific function with a specific user. Users need to be able to specify which functions they want to share and with which user. Moreover, access control needs to be established to only allow authorised user activities to access the shared functions. The second aspect is how to automatically fulfil users activities using the most secure set of shared devices. Searching techniques are required to find the most secure shared devices that can fulfil user activities. These two aspects are addressed in the following sub-questions:

(a) How can users share their devices in a secure manner?

To enable secure sharing, a policy language is required to enable users to express their sharing policies. Existing policy languages mainly consider the environmental context (e.g. location) for access control [160, 198, 193]. These policy languages do not support controlled device sharing at the level of individual device functionality. Therefore, we developed a new policy language to allow device owners to precisely specify the conditions under which they are willing to share their device functionalities. For example, users can specify who can use their devices, which functions can be used, and which devices can interact with their devices.

(b) How can user activity requirements be fulfilled securely without violating user sharing policies?

There is a need for a mechanism to avoid using vulnerable devices to fulfil user activities when more secure alternative devices exist. This problem has been treated as an optimisation problem where the objective function is defined to select the most secure devices that fulfil user activities. We assume that a mechanism exists to quantify an individual IoT device security score (the larger, the better) [105]. For example, a device security score can be derived by inverting the inverse value of the Common Vulnerability Scoring System (CVSS) of a device software [117]. The objective function will be solved subject to a set of constraints that are generated based on the sharing policies.

4. *How feasible and practical is the fine-grained access policy enforcement?*

   This research question relates to the investigation of the feasibility and practicality of enforcing the network and application access control policies that are generated using the proposed methods. In particular, the existing technologies that can be used to implement fine-grained access. A smart space case study will be implemented to enforce network and application policies in various scenarios. The case study will demonstrate the available technologies to enforce network and application access policies and the enforcement overhead.

1.4 Research Objectives

The following research objectives have been defined to fulfil the overall goal of the research questions:

1. *Develop a new security framework to allow users to customise IoT devices and pre-defined policies and integrate this policy with network security middleboxes.*

   The proposed framework aims to achieve collaborative network access control between users, IoT manufacturer policies, and network security boxes to achieve fine-grained access control that mitigates a malicious IoT device threat. The proposed framework has three features: (a) it allows pre-defined policies to be enforced for IoT devices to provide the minimum security. For example, applying a MUD policy to define the manufacturer cloud to which an IoT device can connect; thus, reducing the risk associated with the exposure of IoT devices to the Internet; (b) it enables users to customise and overwrite these policies to meet user security and privacy requirements; (c) it enforces an access policy based on feedback from security middleboxes.

To further mitigate malicious devices threat; two security services will be developed. These services will also show the flexibility of integrating security serviced into the proposed SDN-based framework. In particular, ARP server mitigating the common Address Resolution Protocol (ARP) spoofing attack and another security service to mitigate ongoing attacks dynamically by integrating IDS with the framework.

2. *Develop a new user activity representation and automatically search for IoT devices to fulfil user activities while optimising user preferences.*

   We use a functional workflow to represent user activity functions. This representation initiates the user activity recognition process by defining its functional requirements, which can then be fulfilled by a selected set of devices to meet user preferences. Optimisation algorithms will be used to automatically select the set of devices to fulfil user activities and maximise their preferences, systematically generating access control policies to ensure the principle of least privilege.

3. *Develop a new system to allow users to securely share their IoT devices under precise sharing policies and optimise the usage of the most secure devices to fulfil their activities.*

   To share IoT devices with others, users need a sharing policy language to allow device owners to precisely specify the conditions under which they are willing to share their devices. Users may want to specify who can use their devices, which functions can be used, and which devices can interact with their devices in a sharing environment. A novel IoT architecture is proposed with a newly designed IoT device sharing policy language. By using the sharing language, devices owners can clearly define the circumstances under which their devices can be shared with specific users. To address the optimality aspect, we focus on minimising the risk of using vulnerable shared devices by using the most secure set of devices to fulfil user activities.

Considering secure sharing and secure using aspects, the IoT device sharing problem is mathematically formulated and further transformed into an equivalent Integer Linear Programming (ILP) problem. An integer programming problem is a mathematical optimisation program where some or all of the variables are restricted to be integers. ILP is introduced in 2.1.7 and Section 5.7 explains how it is used to address IoT sharing problem. The aim is to optimise the set of devices that fulfil user activities to the most secure available ones subject to a set of constraints that represent users' sharing policies.

4. *Implement a case study testbed to explore the available techniques to achieve fine-grained access policy enforcement.*

   A smart home space case study is implemented to demonstrate how fine-grained access control can be enforced using existing technologies. A Policy Enforcement Point (PEP) is implemented to enforce access policies in the network and application layers. To demonstrate the feasibility and performance of the PEP, we use a hub-based approach. User activities are deployed in the hub, which controls all other devices. The hub commands pass through a PEP proxy. The PEP is implemented in the network and application layers. In the network layer, the PEP blocks device-to-device communications and only allows the hub and device to communicate. Then, it redirects the hub-to-device connections through the PEP web proxy. In the application layer, the PEP web proxy is implemented to intercept hub-to-device control messages and allow or deny them based on the application-level access policies. A set of experiments are conducted to report the performance of the system, measuring the overhead introduced by the PEP.

Although the thesis did not focus on usability, it does consider it by applying usability principles. For example, manual rules are implemented in the first objective similar to existing home router NAT firewalls. Also, the second and third objectives avoid adding any burden on end-users by deriving access policy automatically without user intervention.

1.5 Major Contributions

A summary of the major contributions of this thesis is presented in this section, and the following chapters (3, 4, 5, and 6) are dedicated to discussing each contribution presented below.

1. This thesis proposes a new access control framework that enforces IoT manufacturer, security provider, and user policies. This architecture enhances the smart space security by utilising SDN to enforce the least privilege access control using MUD and user policies. A smart space IoT testbed evaluation demonstrates that the proposed framework reduces the attack surface.

   This contribution has been published in:

   Al-Shaboti, M., Welch, I., Chen, A., & Mahmood, M. A. (2018, May). Towards secure smart home IoT: Manufacturer and user network access control framework. In 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA) (pp. 892-899). IEEE.

2. This thesis proposes a novel IoT application-centric access control approach utilising user activity automation to define the underlying IoT device interactions and enforce least privilege access control. Hence, the proposed approach does not require user intervention to define the devices to use or the access control to enforce. Device selection is achieved using the user preference model, whereas access control is determined using the IoT device capability description.

   This contribution have been published in:

   Al-Shaboti, M., Chen, A., & Welch, I. (2019, August). Automatic Device Selection and Access Policy Generation based on User Preference for IoT Activity Workflow. In 2019 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE) (pp. 769-774). IEEE.

   Al-Shaboti, M., Welch, I., & Chen, A. (2019, July). IoT Application-Centric Access Control (ACAC). In Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (pp. 685-687).

3. This thesis proposes a secure sharing policy architecture for a multi-user smart space with a newly designed IoT device sharing policy language. By using our language, device owners can clearly define the circumstances under which their devices can be shared with specific users. We focus on minimising the risk of using vulnerable shared devices by maximising the overall security score of devices selected to fulfil user activities considering user sharing policies. We show that any device sharing policies can be precisely represented as hard ILP constraints to completely prevent policy violation. The experimental results demonstrate that the IoT sharing issue can be successfully solved using popular and widely available ILP solvers

such as the CBC MIP and CP-SAT.

This contribution has been accepted at:

Al-Shaboti, M., Chen, A., & Welch, I. Achieving Optimal and Secure Sharing of IoT Devices in Multi-User Smart Spaces. LCN 2020 45th IEEE Conference on Local Computer Networks. (Accepted)

4. This thesis explores the practicality of enforcing access control policies using existing smart space technologies. It presents a case study that demonstrates the available technologies to implement a fine-grained access control smart space framework. In addition, it investigates how the IoT semantic model in the WoT technology can be used to retrieve IoT device capabilities and Application Programming Interface (API) to define automatic fine-grained policies at the application level for each device usage in each user activity.

1.6 Organisation of the Thesis

The remainder of this thesis is organised as follows. The necessary background and related works are reviewed in Chapter 2. Each of the five subsequent chapters, i.e. Chapters 3 to 6, addresses one of the research goals. Chapter 7 concludes this thesis.

Chapter 2 includes two sections. The first one presents the background of the IoT concept, user activity automation, and the relevant IoT vulnerabilities in smart spaces. It also introduces some searching algorithms and optimisation methods that are used in this thesis. The second section presents related research that addresses IoT vulnerabilities in smart spaces for single and multiple users.

Chapter 3 proposes a new smart space IoT security framework that utilises SDN to allow the coexistence of IoT manufacturers, network security services, and user access control to enhance the smart space IoT security. This chapter also presents a approach to mitigate IPv4 ARP spoofing through an NFV ARP server (example of security service), such that all ARP requests are generated by a trusted entity (i.e. ARP server).

Chapter 4 presents a new user activity representation that decouples functional user activity requirements from the underlying devices that fulfils them. This representation allows users to describe their activities in terms of required functions and provides for a dynamic device selection using a newly proposed method. This chapter also presents an automatic network access control policy to enforce least privilege access control on the IoT devices.

Chapter 6 presents a case study that demonstrates the available technologies to implement a fine-grained access control smart space framework. In addition, it shows the overhead of enforcing access control at the network and application levels on user activities. More importantly, it investigates how WoT technology can be used to extract device capabilities and API, which can be used to define automatic fine-grained policies at the application level for each device usage in each user activity.

Chapter 5 proposes a new IoT system for IoT device secure sharing in multi-user smart spaces. The proposed system is supported by a newly designed sharing policy language. The system treats the secure sharing and using as an optimisation problem, in which the goal is to fulfil user activities using the least vulnerable devices subject to sharing policies.

Chapter 7 concludes this thesis and summarises the key findings. The research contributions and key points of this thesis are highlighted and discussed. This chapter also suggests and discusses different opportunities for future works.

Chapter 2

Background and Literature Review

This chapter serves as a foundation for this thesis. It covers essential background (Section 2.1) and provides definitions of the basic concepts and terminologies in smart space IoT, including its architecture, automation, and protocols. The chapter also provides a summary of IoT network vulnerabilities, threats, and existing related network access control technologies. This chapter also introduces a brief introduction to different IoT access control systems. Details about fine-grained access control, Manufacturer Usage Description (MUD), Software-defined Networking (SDN), and optimisation problems and methods that are related to this thesis are also presented in this chapter.

This chapter then reviews related work and summarises the research for the following three contribution chapters 3-5). In particular, Section 2.2 presents the related works in IoT access control systems, automatic activity fulfilment, and multi-user IoT sharing.

2.1 Background

2.1.1 IoT Ecosystem

The IoT ecosystem includes all components that enable end-users to use IoT. These components can be categorised based on the layers they belong to as follows [88, 162]:

1. The perception layer, also called the physical layer, is the bottom layer of the ecosystem. Functions in this layer include identification, sensing, actuating, and communication. It passes data to the network layer.

2. The network layer is the glue between the perception layer and the application layer. It also includes device management and fog computing for local processing.

3. The application layer is the top layer of the IoT ecosystem, where the data is analysed to provide the desired IoT services.

2.1.2 Smart Space IoT Architecture

The IoT smart space architecture consists of four main components. A typical smart space architecture is illustrated in Figure 2.1 and its components are explained as follows:

- The thing; which is the smart device that is associated with some sensors and/or actuators capabilities. The thing should be bound to a user account in the cloud before it can be used.

- User agent software that enables users to control the thing. This can be a mobile app or web browser that gives the user access to his/her IoT cloud account.

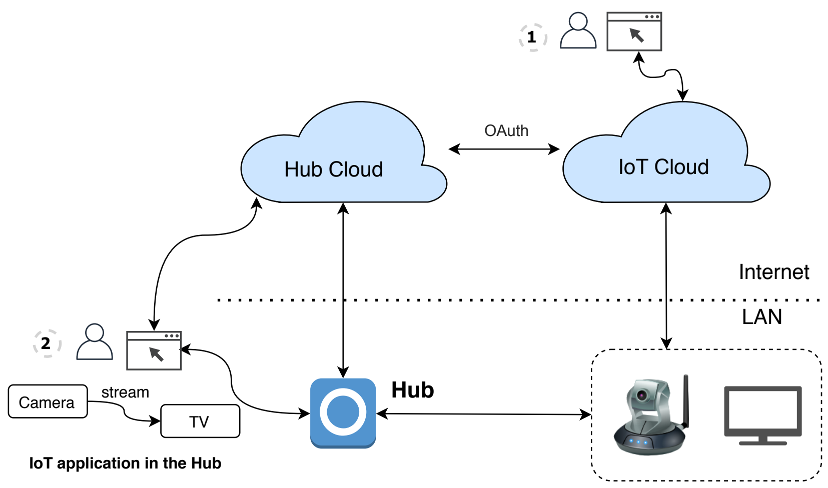


Figure 2.1: Smart space IoT architectures: (1) A single device is used through the cloud; (2) Hub-based automation for multiple devices control.

- IoT cloud services and API that host management, control, and data processing tasks that enable users (through their mobile application agent) to use their IoT things.
- Communication technologies (e.g. Ethernet, WiFi) that connect IoT things.

There are two models to control smart space IoT devices, as shown in Figure 2.1; IoT cloud and IoT hub.

1. **Cloud-based** where physical IoT device status is saved and synchronised with digital twines. The IoT cloud provides APIs for a third party or a mobile app to read and changed the IoT twin status, which works as a proxy for the physical IoT device. Users control their IoT devices individually through the cloud. For example, in this model, users can manage their camera and TV individually, using their own app or web interface. However, this model doesn't fulfil users requirements when it comes to automating activities that require more than a single device.

2. **Hub-based** where all IoT devices are bound to a local central controller, which then controls these devices via local connections or through their cloud APIs. The Hub can be a device such as Alexa and Google Home or software such as OpenHAB and Mozilla-WoT gateway that is usually installed in a Raspberry PI [124, 141]. Hub architecture enables IoT automation using multiple devices. For example, users can stream IP camera video to the smart TV.

Activity Automation

Automation is defined as the "execution by a machine agent (usually a computer) of a function that was previously carried out by human" [143]. IoT automation facilitates various user activities in home, office, building or shared spaces such as a) Home automation such as lighting system, thermostats, door openers; b) Security system such as security cameras, motion sensors, and baby camera; c) Smart appliances like fridge, washing machine, vacuum cleaner etc.; d) Smart entertainment systems such as audio system, smart TV, and PSx stations.

To ease automation process for users, automation platforms utilise flow-based programming [119] paradigm, see Section 2.1.2, such as Mozilla WebThings Gateway [124], NodeRed [142], and Microsoft flow [52]. Other platforms use an integrated intelligent personal assistant that allows users to control their smart home using their voice, such as Alexa [10]. Hub also can be a software platform such as IFTTT (If This Then That) [83] which uses IoT cloud API to send commands and query device status. Hub platforms require authorisation to access by users to their devices IoT cloud services which are often granted using OAuth protocol [76].

Flow-based programming

Flow-based Programming (FBP) is a component-oriented programming paradigm, invented by J. Paul Rodker Morrison, that uses a "data process-

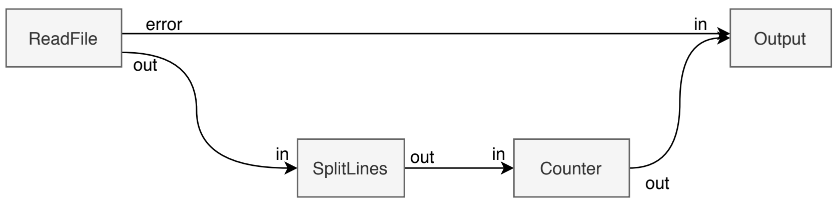


Figure 2.2: Counting number of lines in a file using flow-based programming (reproduces from [19])

ing factory" metaphor for designing and building applications [119]. In flow-based programming, an application is defined as a network of predefined processes as black boxes that have predefined external connections (e.g. in and out interfaces) to communicate data via information packets. These black-box processes can reorganise to form different programs without having to be changed internally. Figure 2.2 shows an example for a flow-based programming; the program counts the number of lines in a file through connecting predefined processes *ReadFile*, *SplitLines*, *Counter* and *Output*.

IoT binding process

Before users can use an IoT device, they need to create cloud accounts and associate them with the physical IoT device, this process is called binding. Through the binding process, the IoT cloud binds the IoT device to the user account. Therefore, the user can send commands to the device and check its status through the cloud. The IoT device and user binding are explained in [29] as follows.

1. Authentication: the user and IoT device authenticate themselves to the cloud, using username, password and DeviceID, respectively.

2. Local binding: the user uses an app to exchange DeviceID and UserToken with the IoT device. This step is necessary to verify that the user physically possesses the device.

3. Remote binding: either the device or the user app send the (DeviceID, UserToken) to the cloud for remote binding. After this step, the user can control the IoT device.

4. Binding revocation: when the user resets the device or deletes the device from the app, a revocation message is sent to the cloud to remove the binding between the user and the device.

Web of Things (WoT) The Web of Things (WoT) is a W3C standard to improve the inseparability and usability of the IoT. It works at the application layer of the IoT and utilises semantic models for thing's descriptions, data, and interactions. Hence, it allows exposing virtual or physical IoT devices as a resource with a description of its capabilities that other web services can consume.

Each thing in WoT is called *servient* which is a term to indicate that a thing works as a server and a client at the same time. Figure 2.3 illustrates the main components of WoT servient which are: a) script that knows how to control the thing as a response to request from consumer (e.g. browser); b) Resource model which provides Uniform Resource Identifiers (URIs) for the thing properties and capabilities; c) Thing Description (TD) which is a semantic description of the thing and its capabilities. TD is consumed by the WoT client to understand what is the Thing and how to interact with it. d) Protocol binding defines the concrete messages that are exchanged during communication.

IoT Protocols

Message Queue Telemetry Transport (MQTT) MQTT is a lightweight publish/subscribe application layer message protocol for IoT [137]. It is designed to enable machine-to-machine connections where a small code footprint and network bandwidth are required. As it uses the publish/subscribe message pattern, it supports one-to-many, one-to-one, and many-to-many

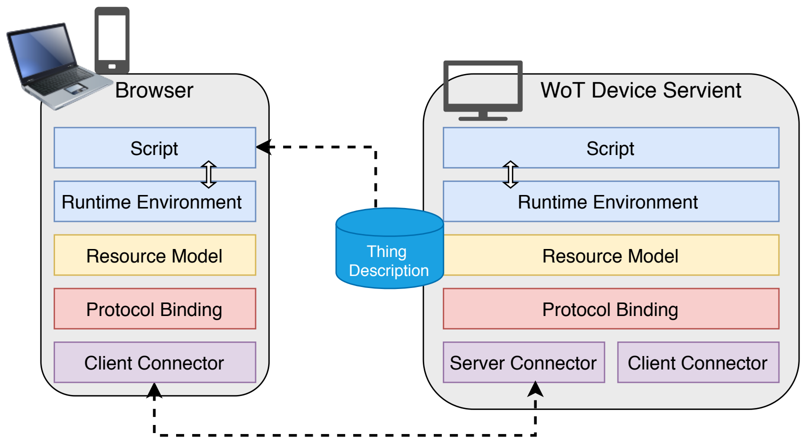


Figure 2.3: WoT Servient for smart TV that are controlled via a web browser.

message distribution and decoupling of applications.

MQTT decouples clients and servers applications such that a client can send messages to a server regardless if the server is running or not. The broker facilitates this feature, who maintains queues for topics and the corresponding publishers and subscribers to each queue. Figure 2.4 shows the MQTT architecture where a motion sensor publisher motion data to motion topic in the broker. The motion topic is then consumed by the light application (the subscriber) so it may turn on the light when the motion sensor detects someone.

MQTT provides the following three qualities of service for message delivery :

- *At most once*, in which messages are delivered, where messages are delivered or not depends on the environment. Therefore, messages can be lost in this level. It is useful where many messages are published within a short inter-arrival time, so if one is missed, another one will be published soon later.

- *At least once*, this level assures that at least one message arrives. How-

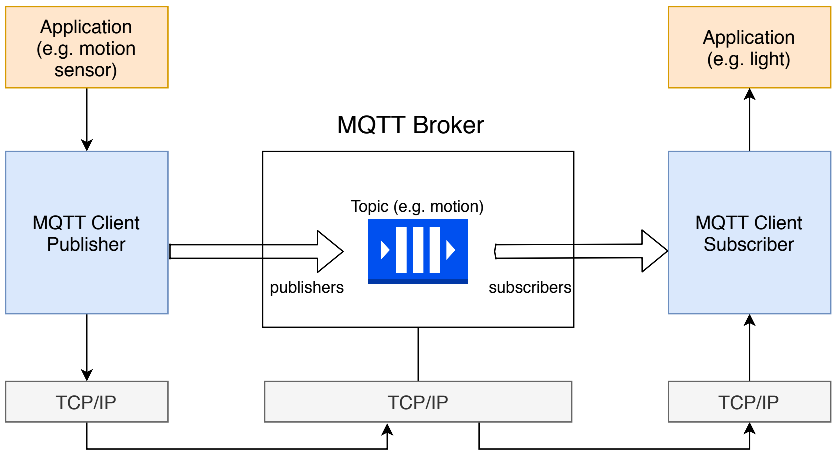


Figure 2.4: MQTT architecture

ever, more than one copy of messages may be received.

- *Exactly once*, where messages are assured to arrive exactly once. It is a firm level where duplication or loss are not allowed.

OAuth 2.0 The OAuth is the main authentication and authorisation protocol that enables IoT automation. It facilitates limited access authorisation to protected resources through an access token rather than the owner credentials, see Figure 2.5. The token is issued to the client by the authentication server after it takes authorisation grant by the resource owner. Communication takes place over HTTP on the top of Transport Layer Security (TLS). **Access token** is a string (credential) used to access protected resources, it includes scope of access, duration, and it may also require authorisation information to be valid.

Initially, the client developer should register its application with the authorisation server, which includes:

1. *Client type*, confidential such as web application where token is stored securely in the server side, public such as browser scripts, or native

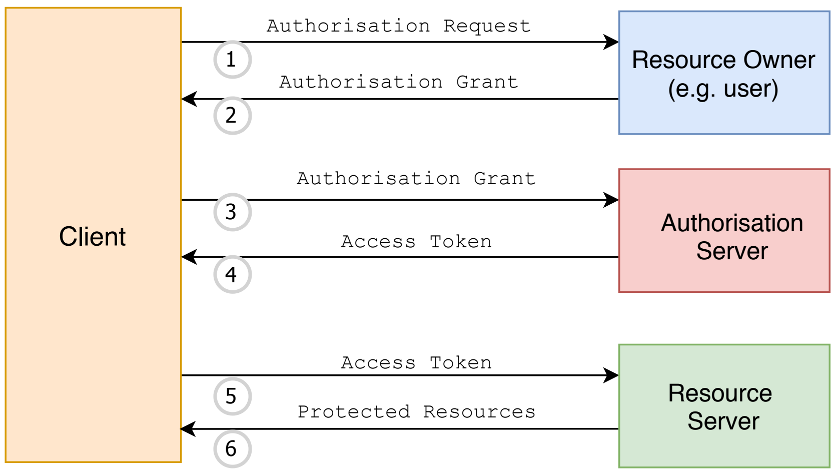


Figure 2.5: OAuth abstract protocol flow (reproduced from [76]).

application/desktop application/android apps that are running on clients devices.

2. *Redirect URI*, defines where to redirect the resource owners after the authorisation server either grants or denies the client's request.

3. *Other descriptive information* such as website name, logo, terms and conditions. The authentication server grants the client an identifier that will be used later to identify the client when user grants it access.

OAuth supports four types of authentication grants based on the client request method: **Authentication code grant** Used for server-side application, where the code is securely stored. It includes the following steps, see Figure 2.6:

- A- When users use the client application and grant it an access to the resources, client redirects the users to the authorisation server along with the client registration information.
- B- Authentication server authenticates the users and checks their grant (allow or deny).

- C- If users grant the access to the application, then they will be redirected to the client redirection URI along with the authentication code.

- D- The client requests an access token by providing authentication code and the redirection URI for verification.

- E- The authentication server checks the request and grants the token or denies it.

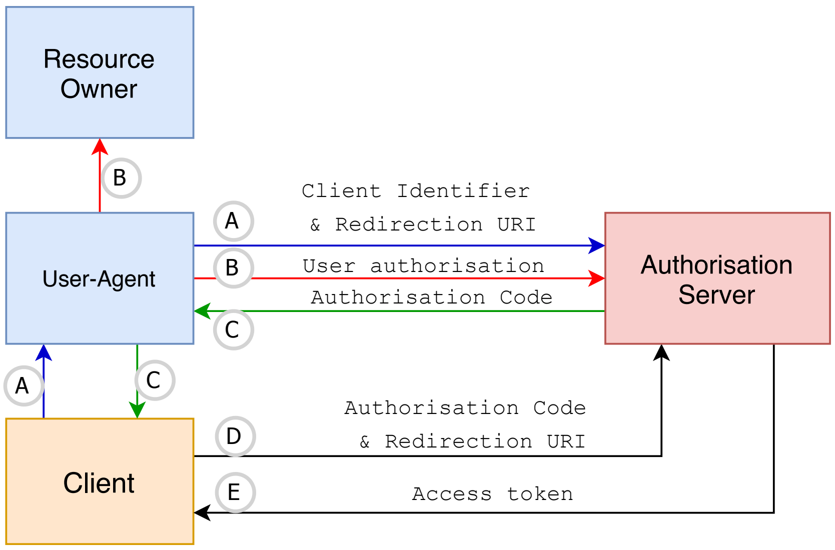


Figure 2.6: Authorisation Code Flow. Steps (A),(B), and (C) pass through the user-agent. (reproduced from [76]).

Implicit grant Used for clients that are implemented in browser script (e.g. javascript. No authentication code is issued; instead, an access token is granted directly to the client. User-agent plays the role of a proxy between the authentication server and the client.

Resource owner password credentials grant User credentials will be used as an intermediate stage to get access token. However, it is only used once, and the client gets a long-lived access token and maybe a refreshed token such that the client does not need to store user credentials. This type of authentication grant must be used only with highly trusted clients.

Client credentials grant It allows client (that represents user owner) to get access to protected resources by the client control.

2.1.3 Software-defined Networking

Software-defined Networking (SDN) is a promising technology to tackle dynamic access control enforcement. Network management used to be a complex task in which high-level policies had to be compiled into various low-level configurations based on network device vendor. Network researchers realised that the problem lies in the architecture of the network itself, more specifically due to the coupling between the control plane and the data plane. They experienced difficulties in proposing new network function/protocol and increased complexity with every newly added network feature.

There were many attempts to separate control plane from data plane such as SANE [25] and Ethan [24]. Moreover, OpenFlow was developed to unified the communication between the control plane and the data plane. Lately, these two features (decoupling control plane from the data plane and unified control and management protocol for all network devices) formed what we call software-defined networking (SDN) [47]. SDN decouples the control plane from the data plane and allows the programmability of the data plane. Such that a data plane becomes a forwarder and is controlled by a control plane through a southbound API such as OpenFlow. Moreover, a control plane is programmable through northbound API. This fundamental change in the network architecture empowers SDN with a set of desirable features: dynamic flow control, network-wide visibility, network programmability, and a simplified data plane.

SDN proliferates in industry as well as in academia as it reduces the production time of network devices and also allows researchers to test

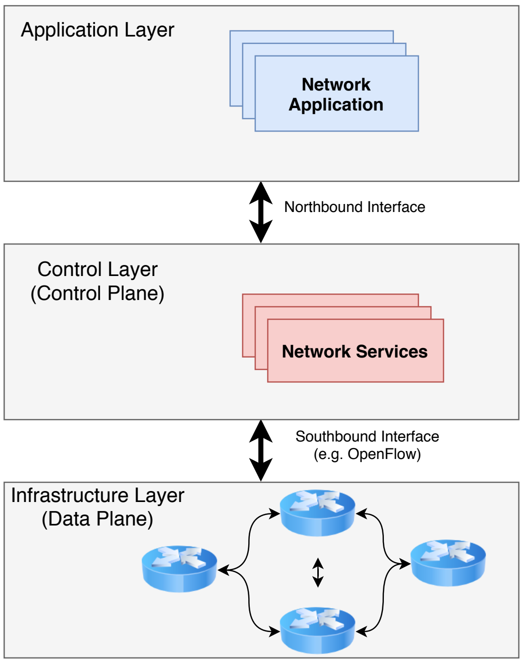

Application Layer
Network Application
Northbound Interface
Control Layer (Control Plane)
Network Services
Southbound Interface (e.g. OpenFlow)
Infrastructure Layer (Data Plane)

Figure 2.7: SDN architecture

new proposed solutions either in an emulated environment (e.g. Mininet) or even in parallel with an operation network (e.g. via FlowVisor[163]).

OpenFlow

The main idea of OpenFlow was to enable researchers to run experimental protocols in the same production network [114]. It comes with a new paradigm of networking (i.e. SDN), in which control decision is made in a separate entity (the controller) left switches without any control decisions. OpenFlow specifies the protocol of communication between the controller and the switch and the switch components and functions to enable OpenFlow.

Typically, controller manipulates OpenFlow switch's flow-tables (built from TCAMs) through secure channel using OpenFlow protocol see Figure 2.8. Each flow entry consists of match, auction, and statistical parts; a switch matches each packet header with the flow header part. If a match exists, then the switch executes the auction parts and updates the statistics for that particular flow (i.e. meter table). OpenFlow switch can match ten packet's headers (In port, Ethernet (source, destination, type), IP (source, destination, protocol), TCP (source, destination)). It also supports drop, encapsulate, and forward actions.

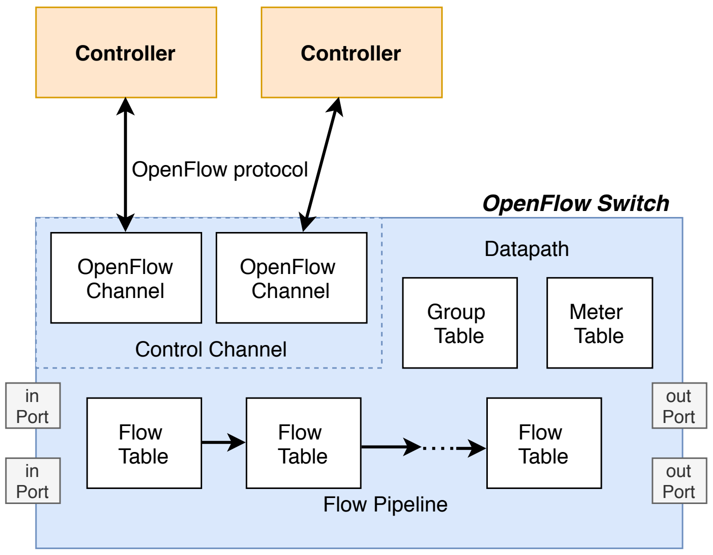


Figure 2.8: Main component of OpenFlow switch (reproduced from [140]).

2.1.4 IoT Vulnerabilities

The Open Web Application Security Project (OWASP) lists the top 10 attack surface areas in the IoT ecosystem [138] which include insecure web interfaces, insufficient authentication/authorisation, insecure network services, and lack of transport encryption. An HP report shows 70% of IoT devices used unencrypted network protocols, and 60% of IoT user interfaces are vulnerable [46]. Consequently, hackers can launch different types of attacks ranging from eavesdropping to completely taking control of the IoT device. Jacobsson et al. [86] applied Information Security Risk Analysis (ISRA)[145] on Smart Home Automation System (SHAS) with respect to security and privacy. Analysis conducted by security experts, domain experts, and system developers show that out of 32 examined risks, nine were low, four were high, and 19 were moderate risks. In 2017, an Arbor report estimated the current vulnerable IoT devices to be 10 billion [132]. A systematic evaluation of privacy and security for consumer IoT devices, presented in [105], shows that the majority of threats are due to vulnerabilities in access control.

Some of the key reasons behind IoT vulnerabilities are summarised as follows [95, 106, 173]:

- Some vendors are new to cyber security and are not aware of what can go wrong, since their main expertise is in lighting, electronics, air conditioning, etc., but not security [95, 106]. Moreover, the capabilities of the IoT are limited, especially when it comes to encryption, as it requires additional computational power.
- IoT is an emerging technology with no regulations that enforce security as a requirement for manufacturing IoT devices [159].
- Many of the IoT devices are cheap and vendors cannot afford deep vulnerability analysis for their products.
- Users often do not update their IoT firmware in a timely manner, and vendors do not provide an automatic security update [175]. Also, the short lifetime of the IoT makes updates mechanisms harder. Moreover, some IoT vendors go out of the market, which left their devices without support.
- Fragmented IoT supply chain makes it difficult to form a partnership between different IoT market players (e.g. components, tools, developers, supporting services suppliers) [95, 175].
- Diverse standards and technology make designing a secure product more challenging.

- IoT is built with a specific purpose and powered by embedded systems with relatively limited resources that make it impossible to apply the conventional security solutions [61, 167].

2.1.5 IoT Access Control

Previous research in securing IoT can be classified into network perimeter-based and network-wide based. Each of which can be classified into fully automated or considering user input as follows.

The network perimeter-based solution in which network policy is enforced at the boundary of the network [40, 192]. Xu et al. [192] proposed a bloom-filter based framework to analyse home network inbound traffic to detect persistent threats. On the other hand, DeMarinis et al. [40] predict the benign IoT traffic with the aim to filter unwanted traffic using an access policy. Others delegate security to the ISP to be enforced in the edge switch [169]. However, network perimeter-based solutions cannot provide protection from an internal malicious device. For example, any compromised device can scan the network internally and open ports for an external attack as in [168].

Network-wide access control can enforce fine-grained access control and can be enforced across various types of IoT devices. It can protect against a malicious connection that comes from outside the network or between devices within the network. For example, a compromised device can scan the network internally and open ports for an external attack as in [168]. The current trend is to treat internal and external hosts as untrusted. Examples of such solutions are Software-defined Perimeter (SDP) [57], and Google BeyondCorp security framework [188].

The automation-based solutions automate security detection and prevention mechanisms often without considering user input. Yu et al. propose a security policy based on IoT signature and anomaly behaviour and use a micro Virtual Machine (VM) that acts as a proxy for each vulnerable IoT [196]. However, deploying and configuring a VM gateway for each IoT device is going to introduce an extra overhead. Similar work also proposed by [108], in which an agent is integrated with IoT devices and controlled by the SDN controller. There is ongoing work by IETF to form a Manufacturer Usage Description (MUD) standard [101]. MUD would allow manufacturers to specify the least access privilege necessary for the IoT, which will be enforced by the network management system. Consequently, it prevents unintended/malicious connections.

The user-defined policy solutions can be classified into four subcategories: a) QoS control which allows the user to manage and prioritise their application Quality of Service (QoS) [97, 64]. For example, the user may give high priority for an ongoing Skype conference over other connections; b) Internet data cap management [30] such that users can set cap settings for each device and monitor their data cap usage; c) Host-based firewall which controls network access to/from individual devices through a personal firewall (e.g. iptable in Linux) [150]; d) Home gateway firewall to control network access between trusted (internal) and untrusted (Internet) networks. For example, users can set firewall rules to restrict a device's internet access to port 80/443 using a router firewall [183]. This approach becomes less effective with the increasing number of IoT and mobile devices inside the trusted network. Such devices, if compromised, can become entry points to compromise the rest of the network without going through the firewall. This category shows that in the home network context, there are already management and security solutions that enable users to enforce their security decision on the device level, and to some extent at the network level (e.g. home gateway firewall). However, it primarily addresses the external connections and does not provide users with any control over their own network. For example, users may want to allow their guests to access the Internet and the lighting system but not the CCTV system.

Discretionary Access Control (DAC) at the network level has not been fully explored in the existing smart-home IoT research. Studies mainly concentrate on two approaches either fully automated security decisions as in the second category or delegate it to a third party as in the first category. The existing network DAC is mainly based on network perimeter (e.g. home router firewall), which is not efficient in the smart home environment where there are many mobile devices and vulnerable IoT inside the network.

Principle of Least Privilege

The principle of least privilege is one of the main security principles, also known as the principle of minimum privilege. It is introduced in 1975 by Jerome Saltzer and Michael Schroeder's and it specifies that every process should not be given any more privileges than absolutely necessary to do its job [158]. By reducing the number of privileges given to a process, the principle of least privilege minimises the number of ways a process may abuse its privileges.

For enforcing the principle of least privilege, a fine-grained access control mechanism is required. If a system doesn't have fine-grained access control, then a process will get privileges more than what it requires to do the job.

Another concept related to and complements the least privilege is the minimisation principle. It states that the software must be developed to do the intended job and no more.

Manufacturer Usage Description (MUD)

In contrast with general-purpose computers, IoT devices are usually designed with a purpose and intended limited function. Therefore, MUD comes to avoid using these devices in unintended activities by adversaries. Basically, MUD is a file that introduces a device to the network and tells what and what, not this device is intended to do on the network [101].

The MUD will help to reduce the attack surface by limiting the access from/to the device. It also provides a sort of protection for vulnerable devices.

MUD defines three requirements:

1. A device classifier which is used to locate the device description.

2. A device description and how it is interpreted. This component is represented by Yang-based XML or JSON file.

3. A means to retrieve the description.

A smart light bulb can be controlled remotely by smartphone apps to light a room. However, it does not need to access a printer or Facebook and so many other internal and external services. Therefore, the MUD will provide a level of protection to the light bulb as well as to the rest of the network by allowing it only to access what is required for its function only.

The MUD URL A device emits a URL that classifies the device and provides a means to locate the device file description (policy file). There are three proposed ways for a device to emit the URL; DHCP option, X.509 extension, and LLDP extension.

- The MUD URL DHCP option consists of code, length, and MUD URL. Code option is assigned by IANA for IPv4 and IPv6. Len is in octets and must not exceed 255. The DHCP client has only one MUD URL at most, and if the DHCP server processes the MUD option, it responses with a MUD option with len=0. If the DHCP server does not process the MUD URL but forwards it to the network management entity, then it has to notify it of any changes in the DHCP state of the client (e.g. lease expires).

- The MUD URL X.509 extension contains a single URI for an online MUD. This extension can be used in the manufacturer certificate 802.1AR (IDevID) or deployment certificate (LDevID).

- The MUD URL Link Layer Discovery Protocol (LLDP) extension is a one-hop local area network protocol used by end devices to advertise their identities, capabilities, and neighbours. LLDP is a Type-Length-Value (TLV) protocol that uses an Organisationally Unique Identifier (OUI) with the vendor-specific TLV (type=127) to be used for advertising MUD URL.

Techniques for Enforcing Access Control

Access Control List (ACL) approach is a common way for access control in the IoT network [18]. The capability-based model assigns rights to subjects prior to its current context [18]. However, in ACL model, access is granted based on attributes/context that might be static or changed dynamically. Moreover, an ACL model can be enforced at the network level. This has the benefit that there is no need to install special libraries on clients such as IoT devices [5]. Common ways of enforcing access control are as the following:

Software-defined Networking

SDN mainly opens two security research directions: 1) Securing SDN itself, 2) Enhance network security due to its desirable features such as dynamic flow control, network-wide visibility, network programmability, and simplified data plane [164]. However, most of the work has focused on the second direction, where they utilised SDN to provide security solutions [7] such as security configuration, threat detection, threat remediation, and network verification.

The following SDN properties motivate employing SDN for dynamic IoT access control in a smart space.

- *Dynamic flow control.* With the help of SDN, we can dynamically control network flows. For example, by employing SDN, we can easily implement a network-wide firewall that not only controls traffic between two network segments but also between the individual devices in each segment. These powerful and rich SDN functions enable dynamic flow control [15].

- *Network programmability.* A programmable SDN controller can control multiple data planes (e.g. router, switches). This programmability allows us to create dynamic access control as an application, reducing the efforts required to develop, maintain, and update an access control application.
- *Separation of concerns.* Separating the control plane from the data plane allows for a more modular network where switches can be updated with a minimum update in the control plane.
- *Network-wide visibility.* The controller communicates with the data plane using compliant protocols which enhance network visibility. All data planes are connected to a logically centralised control plane that collects network status information. Therefore, a network application running on the control plane naturally has a view of all connected data planes, and it can control them in a centralised way.
- *Scalability.* SDN paradigm is scalable enough for smart spaces. Google has deployed SDN for datacenter backbone traffic and reports network utilisation of 95% [89].

Attracted by its features, researchers used SDN to enhance network security [164], and developed security solutions under SDN, [108, 103, 169]. SDN has a set of desired properties that makes it attractive for security researchers, including dynamic flow control, network programmability, separation of concerns, and network-wide visibility. The security research community have developed security solutions under SDN such as Resonance, OF-RHM [87], and NetFuse. Moreover, security frameworks are proposed, such as FRESCO, Procera, FortNox. However, the majority of previous works focus on securing campus/enterprise networks and some are general solutions [42]. Other researchers have utilised SDN for developing IoT network security solutions [72].

Few of the network security solutions are dedicated to home networks, which left home networks far behind other networks in terms of security and management. Among the attempts to understand the behaviour of home networks was the BISmark project [177] which monitors and collects continuous measurements that can help to understand the characteristics of broadband access networks through deploying dedicated home gateways. From the early works, Yiakoumis et al. [195] proposed a solution that allows home users to prioritise their network applications. Users use a user agent that forwards user choices to the ISP to enforce it at the last-mail link and manage traffic inside the ISP network. More recently, Gharakheili et al.[64] proposed a framework to use SDN for dynamic resource reservation, whereby ISP exposes API to a content provider to enable fast/slow lanes to enhance QoS for delivered content.

Recently Xu et al.[192] proposed a bloom-filter based analytic framework to identify persistent attacks towards home networks. Authors deployed programmable routers to collect data over 18 months. By analysing the data, they found that bloom-filter was able to detect long-term distributed attacks. They consider any incoming flow initiated from the Internet as unwanted traffic (e.g. Scanning, backscatter, self-propagated worms activities).

Proxy A proxy is a server between the client and the server. There are two main types of proxies forward proxy (also known as the proxy) and reverse proxy [34].

Forward Proxy. A forward proxy is a server that sits in front of client machines. It intercepts all clients' requests to the Internet and communicates with the origin web server on behalf of the clients. Figure 2.9(a) shows a forward proxy in front of two clients.

A forward proxy is used for several reasons, including the following:

- To implement institutional browsing policies. For example, schools and universities may use proxy to filter web sites students and staff should not browse from within the institution's network.
- To bypass state and institutional browsing restrictions. A proxy can also be used to bypass browsing restrictions as the client request always goes to the proxy and not to the forbidden site directly.
- For privacy reasons. Users may not want their ISPs or the governments to know what they are browsing and they want to keep their activities anonymous.

Reverse Proxy. A Reverse proxy is a server that sits in front of one or more servers. It intercepts clients' requests to these servers and creates requests to the server on behalf of the clients. It is noteworthy that while the forward proxy sits at the clients' edge of the network, the reverse proxy sits at the server edge of the network, see Figure 2.9. Internet and communicate with the origin web server on behalf of the clients. Figure 2.9(a) shows a forward proxy sets in front of two clients.

A Reverse proxy is used to provide several benefits, including the following:

- Load balancing- A reverse proxy can be used to distribute a load of millions of requests to a website to different servers at the server-side. Hence, it prevents any single server from overloading.
- Attack mitigation- A reverse proxy can be used to hide the origin server IP address; Hence, some attacks such as DDoS will hit the reverse proxy instead of the intended server.
- SSL encryption- In case the origin server has limited resources and cannot afford encryption, the reverse proxy can be configured to encrypt all connections to the clients while using unencrypted communications with the origin server.

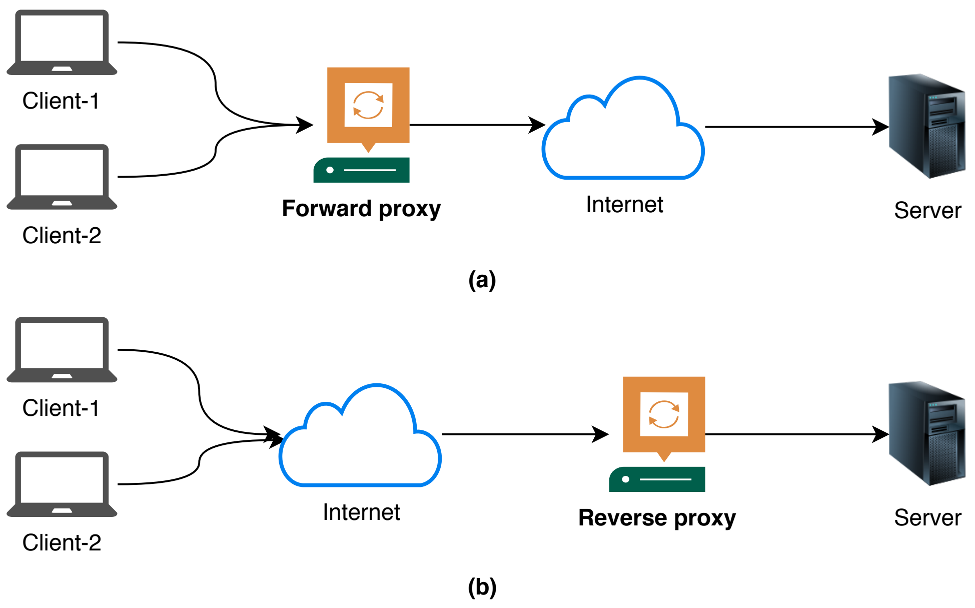


Figure 2.9: Proxy flow; (a) Forward proxy, (b) Reverse Proxy

2.1.6 Bayesian Networks

Bayesian Network (BN) is a probabilistic graphical model that compactly represents the joint probability distributions via conditional independence. It is compacted because the complete number of joint probabilities that need to be computed is exponential to the number of variables (e.g. $2^n - 1$ for n binary random variables) [39] as in the equation below.

$$P(X_1, X_2, ..., X_n) = P(X_1|X_2, ..., X_3)P(X_2, ..., X_n)$$

The second term, $P(X_2, ..., X_n)$, needs to be expanded again the same way. However, if X_1 is independent of the rest of the variables $X_2, .., X_n$ given X_2 (conditionally independent of the rest of the variables), then this will simplify the first term to be.

$$P(X_1, X_2, ..., X_n) = P(X_1|X_2)P(X_2, ..., X_n)$$

We still need to expand the second term, but if we can find conditional independency among the rest of the variables, then this will simplify until we get a product of a set of conditional probability terms. Then, if we need to construct the joint probability, we only need to specify the number of conditional probability distributions that compactly represents the required joint probability. The less dependency, the more compact form we get to represent the joint probability.

BN consists of a directed acyclic graphical model and a set of probability distributions. A graph $\mathcal{G}$ is represented as a pair (V, E), where V is a set of vectors and E is the set of edges between these vectors. If the set of edges E is directed, we then call $\mathcal{G}$ a directed graph. If a directed graph has no cycles, we call it a Directed Acyclic Graph (DAG).

BN is parameterised using Conditional Probability Distributions (CPD). Each random variable in a BN has a CPD associated with it. The edges encode dependency statements between the variables, where the lack of an edge between any pair of variables indicates conditional independence. Each node encodes a probability distribution, where root nodes encode univariate probability distributions and inner/leaf nodes encode conditional probability distributions. Therefore, a Bayesian network is defined as $(\mathcal{G}, P)$ such that.

1. A Directed Acyclic Graph (DAG) $\mathcal{G}$ whose nodes are random variables.

2. A joint probability distribution P of the variables in $\mathcal{G}$.

3. Each node is conditionally independent of its nondescendents given its parents (i.e. Markov condition).

Meaning if $\mathcal{G} = ((V, E), P)$ is given and each node is conditionally independent of its non-descendants, given its parents. It is then said that G satisfies the Markov condition with P, and that $(\mathcal{G}, P)$ is a Bayesian network. Then instead of writing the joint probability as

$$P(v_1, v_2, ..., v_n) = P(v_1|v_2, ..., v_n)P(v_2|v_3..., v_n)P(v_3|v_4..., v_n)P(v_4..., v_n)$$

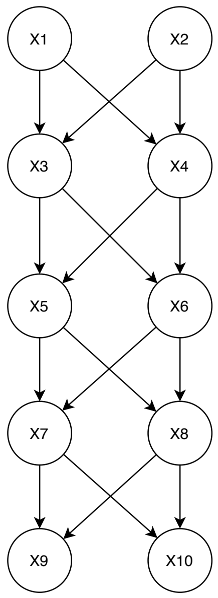


Figure 2.10: Bayesian network DAG for 10 binary random variables.

It can be written as following:

$$P(v_1, v_2, ..., v_n) = P(v_1|Pa(v_1))P(v_2|Pa(v_2))..P(v_n|Pa(v_n)) = \prod_{i=1}^{n} P(v_i|Pa(v_i))$$

Where $Pa(v_i)$ is the probability of v_i parents. More importantly the reverse is true.

Theorem If (G, P) is a Bayesian network, then P (joint probability distribution) is the product of its conditional distributions in G (i.e. P can be represented by those conditional distributions). Therefore, BN can compactly represent a *joint probability distribution* of many variables. For example, if we have binary random variables $n = 10$, then in order to compute its joint probability, we need to compute 2^10. However, using BN (which exploits the conditional distributions), only 36 values (4*8+2*2) need to be computed, as illustrated in Figure 2.10.

In other words, BN is a probabilistic graphical model $< DAG, P >$ where DAG is a directed acyclic graph of nodes representing random variables $(X_1, X_2, ..., X_n)$ and the edges represents the dependencies between them with probability distribution P. Bayesian network simplify the joint probability of a set of variables to be based on their parents, as following:

$$P(X_1, X_2, ..., X_k) = \prod_{i=1}^{k} P(X_i|pa(X_i))$$

where $pa(X_i)$ is the set set of X_i parents.

This means that if we specify a DAG — known as the *structure*—and conditional probability distributions for each node given its parents—known as the *parameters*— we have a Bayesian network, which is a representation of a joint probability distribution [39].

2.1.7 Optimisation

Optimisation means the best of something, and when optimising something, it means making it the best. Based on the problem nature, the best could be to find the minimum value (minimisation) or the maximum value (maximisation). Mathematical optimisation is a branch of applied mathematics to solve optimisation problems.

There are three basic terms for optimisation problems:

- The objective function, $f(x)$, which represents the output that is being optimised (i.e. to minimise or to maximise).

- Variables, $X = \{x_1, x_2, .., x_n\}$, represent the inputs that you can control (give different values). Variables can be discrete (only have integer values) or continuous.

- Constraints, which are the conditions that limits the values that can be assigned to the variables X. Constraints can be equality constraints or inequality constraints.

There are different types of the optimisation problem:

- Based on the variables: it is classified into continuous and discrete; if a problem variables only make sense if they have discrete values (e.g. subset of integers), then this problem is discrete optimisation. On the other hand, if the variables can take any values, then this is a continuous optimisation problem.

- Based on the constraints: it is classified into constrained and unconstrained optimisation problems; if there are constraints on the variables. For example the sum of variable x_1 and variable x_2 must be less than 10, $x_1 + x_2 <= 10$, then this is a constrained optimisation problem. Based on the nature of the constraints, the optimisation problem can be linear or non-linear.

- Based on the objectives: it can be one objective or multi-objective optimisation problem.

- Based on the data: it can be deterministic if the data are known accurately or stochastic if it is not due to measurements error or prediction data.

A general optimisation problem can be defined using the following notations [11]:

$\mathbf{x} \in \mathbb{R}^n$:	vector of decision variables $x_j, j = 1, 2, .., n$
$f : \mathbb{R}^n \rightarrow \mathbb{R}\{\pm\infty\}$:	objective function
$X \subseteq \mathbb{R}^n$:	ground set, such that $\mathbf{x} \in X$
$g_i : \mathbb{R}^n \rightarrow \mathbb{R}$:	constraint function defining restriction on $\mathbf{x}$:

$g_i(\mathbf{x}) \geq b_i,$	$i \in \mathcal{I}$ (inequality constraints)
$g_i(\mathbf{x}) = d_i,$	$i \in \mathcal{E}$ (equality constraints)

Where the set $\mathcal{E}$ denotes the indexes of equality constraints and $\mathcal{I}$ is for the indexes of the inequality ones.

Let $b_{i\in\mathcal{I}} \in \mathbb{R}$ and $d_{i\in\mathcal{E}} \in \mathbb{R}$ represent the right sides of constraints. The optimisation problem can be formulated as follows:

$$\begin{aligned}
&\underset{x\in S}{\text{minimize}}\, f(\mathbf{x}) \\
&\text{subject to} \\
&g_i(\mathbf{x}) \geq b_i \qquad i \in \mathcal{I} \\
&g_i(\mathbf{x}) = d_i \qquad i \in \mathcal{E} \\
&\mathbf{x} \in X
\end{aligned}$$

Note that if the problem is maximisation, then the sign of f needs to be changed.

The definition above can be used to define some of the optimisation problem as follows:

Linear Programming (LP) where the objective function is linear as follows:

$$f(\mathbf{x}) = c^T\mathbf{x} = \sum_{j=1}^{n} c_j x_j, c \in \mathbb{R}^n$$

and the constraints functions are :

$$g_i(\mathbf{x}) = \mathbf{a}_i^T\mathbf{x} - b_i, \mathbf{a}_i \in \mathbb{R}^n, i \in \mathcal{I} \cup \mathcal{E}$$

and the ground set is $X = \{\mathbf{x} \in \mathbb{R}^n | x_j \geq 0, j = 1, 2, .., n\}$

Non-linear Programming (NLP) where some functions f, g_i are non-linear.

Integer Linear Programming (ILP) where all variables must have integer values. $X \subseteq \mathbb{Z}^n$ or it can be binary if $X \subseteq \{0, 1\}$.

Optimisation Techniques

This section presents four optimisation techniques used in this thesis: hill-climbing, simulated annealing, genetic algorithm, and integer linear programming.

Hill Climbing

The hill-climbing search algorithm works through a loop over all movements in the direction of growing value (i.e. uphill). It terminates when it reaches the highest value, where no neighbour has a better value. Hill climbing, also called greedy local search, does not look ahead beyond its current direct neighbours. It progresses fast towards a better solution. However, it suffers from being trapped at local maximum and flat maximum [156].

Simulated Annealing

The simulated annealing algorithm avoids sucking in the local maximum, which the hill-climbing algorithm suffers from, allowing for moves toward states with a lower value. It does not move uphill and downhill purely in a random way, which is going to be inefficient. Simulated annealing combines a random walk and hill-climbing to be efficient and to be complete (to find a solution if there is one).

Simulated annealing takes its name from the annealing process in metallurgy that is used to harden a metal by heating it to a high temperature then slowly cooling it. This process allows metal atoms to reach a desired crystalline state. Similarly, simulated annealing mimics this process by replacing a current solution S with a new solution R based on a probability P. The probability is given by a sigmoid function $P(R, S, T)$ driven by the difference between S and R (i.e. $\Delta E = (Quality(R) - Quality(S))$ and also a temperate parameter T as in the following equation.

$$P(T, R, S) = \frac{1}{1 + e^{\frac{-\Delta E}{T}}} \tag{2.1}$$

The ΔE, at first, tends to accept to move to a new solution, with high probability (random walk). Then, as time passes, it tends to move to the new solution if it is better than the current solution with high probability and a worse solution with low probability.

Genetic Algorithm

A genetic algorithm works by generating a new state using two parent states rather than by modifying a single state as in simulated annealing or hill-climbing search. Genetic algorithms belong to the Evolutionary Algorithms (EA) class. They use the term individual to refer to a single state and a population to refer to a set of states. An array of bits represents each individual.

GA starts with a random population that may contain hundreds or thousands of individuals (solutions). The initial random individuals' generation aims to allow different types of solutions to be represented in the first population. Each individual is evaluated using the fitness function, which returns a large value relative to the goodness of the individual. The next generation is then produced using a set of the best pairs with high probability and some random pairs with low probability. These pairs then go through a crossover process and mutation process such that each pair generates one new individual in the new generation. Crossover works by randomly swapping a set of bits between parents individuals. In comparison, mutation changes bits in the new individual in a random way.

Integer Linear Programming

Integer Linear Programming (ILP) conveys optimisation of a linear objective function subject to a set of linear constraints over integer variables [35]. It is named integer to differentiate it from the continuous linear programming where the decision variables can have continuous values. The integer programming problem has three variances: pure integer program when all decision variables are integers, a mixed integer program when some but not all, decision variables are integers, and a binary integer program when all decision variables are binary variables.

Integer programming problem can be formalised as in equation (2.2).

$$
\begin{aligned}
&Maximise \quad && \mathbf{c}^T\mathbf{x} \\
&\text{subject to} \ : && \\
& && \mathbf{A}\mathbf{x} + \mathbf{s} = \mathbf{b} \\
& && \mathbf{s} \geq \mathbf{0} \\
& && \mathbf{x} \geq \mathbf{0} \\
& && \mathbf{x} \in \mathbb{Z}^n
\end{aligned}
\tag{2.2}
$$

where $\mathbf{c}$, $\mathbf{b}$, and $\mathbf{x}$ are vectors and $\mathbf{A}$ is a matrix, and all variables are integer.

2.1.8 Summary

The basic concepts and terminologies of IoT and smart space have been reviewed in this chapter. The IoT vulnerabilities, threats, and access control were presented in this chapter. This chapter also presented a brief overview of reverse and forward proxy concepts, Bayesian networks, and the least privilege principle. Moreover, optimisation problems and techniques were also introduced.

2.2 Literature Review

This section reviews the literature for the three following contribution chapters (i.e. Chapter 3-5) as follows:

2.2.1 User Role in IoT Access Control

Based on user role in access control, there are two main categories of research in IoT access control. The first research direction focuses on the technical side of automatic access control and ignoring users input. Motivated by uplifting any security input from users, previous works focus on proposing new solutions that tackle the security without involving users.

For example, IETF proposed Manufacturer Usage Description (MUD) to define network requirements policy that defines explicitly what legitimate connections an IoT device can make [101], see Section 2.1.5. Previous research has investigated generating IoT policy (i.e. similar to MUD policy) automatically without relying on the manufacturers [73, 17, 172]. The proposed approaches generate an access control profile for IoT devices by monitoring their benign traffic during the learning phase and then enforcing it in the deployment phase. Others combined MUD with an Intrusion Detection System (IDS) to mitigate threats against IoT [72] while some only rely on IDS to detect and mitigate threats using SDN [196, 98, 20]. Other research attempted to simplify the interactions between IoT devices in order to reduce the complexity of IoT networks and simplify access control. Following this approach, S. Goutam et al. [70] proposed an approach that categorises IoT devices into controllers (e.g. Hubs) and non-controllers. It allows local cross-device connections only between controllers and non-controller and prevent it between non-controllers devices.

The second category of research used some form of user input to derive IoT access control [131, 144, 92, 104, 193, 185]. For example, in [92], authors integrated RBAC with social network services and allowed end-users to define personalised access policy to govern device sharing. Others proposed methods that allow users to specify their safety policy [104] and physical security policy [193]. Neto et al. [131] proposed user data in an attribute-based authentication system for IoT device life-cycle. Trimanada et al. [185] proposed Vigilia, a device-based network access control system that uses IoT app (e.g. SmartThing App) code to derive access control policy. It then asks the user to configure the concrete device instances that they are using to enable appropriate policy enforcement. Moyano et al. [120], proposes a user-centric SDN management architecture that includes giving users manual access control on the network access for devices in their network.

The predefined policy can not be fine-grained enough to capture all types of interactions between IoT devices. This is because the connections between the devices depend on the user activities they are running, which can not be defined beforehand. Moreover, locally monitoring and learning IoT devices interaction [73, 196, 17, 172] may not exhaust all devices functionality set during the observation period, or the behaviour may change following a change in the automated activities, which requires re-training.

Recent studies show that users are interested in contributing to their smart space security. Haney et al. [75] interviewed 40 smart home users, out of which 10 wanted more control over their devices. In this research, one user said "I'd like to have the ability to potentially allow or disallow the functionality of all these devices, maybe at given times. I'd like to be able to define what are allowable communications or protocols".

Motivated by [72, 102] Al-Shaboti et al. [5] have proposed a new IoT framework that integrates existing security services such as IDS and enables users to customise pre-defined policies through manual access control rules. They tested the validity and the feasibility of their framework using a smart home IoT testbed which shows its usefulness in reducing the attack surface through enforcing user-defined fine-grained access rules.

2.2.2 IoT Access Control and User Preference

Users prefer to use a different brand of IoT devices based on various factors that include security [75], privacy [199], and quality of the devices. Therefore, user preference implicitly includes user security and privacy requirements. With the increase in the amount of data collected by IoT devices, users are concerned regarding their privacy and data security. They express their concerns about their preferences to use these IoT devices in a different context. A recent study [125] shows that users have different preferences on various IoT when used in various scenarios, and this preference can be modelled and predicted. Bayesian networks [39] is reported to be an effective approach to build user preference model

[153]. Context-aware access control has been investigated by many research [147, 18, 91, 8]. For example, in [147, 8] authors proposed a proactive rule engine that activates access control rules based on context conditions. However, there is a lack of research on IoT access control that considers user preference.

User Activity Representation

User activity is often represented using a flow-based programming [19] which defines the activity as a workflow of wired processes representing IoT devices and communicate through passing messages between them. This representation is common and supported by many IoT frameworks such as NodeRed [142], Mozilla Gateway [121], and Microsoft flow [52] to realise network activity automation. The concept of activity automation and flow-based programming is introduced in sections 2.1.2 and 2.1.2.

Researchers used activity workflow to optimise network performance [178, 65] and to check unintended actions [187, 193, 196, 104, 126, 27, 28]. Users activities are used to optimise latency and reduce bandwidth and utilise fog computing to enable distributed processing of activity workflow [178, 65]. Other research used automated activities to check unintended vulnerabilities due to the complexity of having many activities [187, 193, 196]. Wang et al. [187] analysed user activities to identify their security risks due to inter-activity vulnerability. Others [193, 196, 104] used policies to prevent such vulnerabilities when users create their activities proactively. Liang et al. utilised IoT workflow to ensure safety by verifying that any flow must never violate the safety policy [104]. Authors in [126, 27, 28, 43] proposed analysis systems that check workflows to validate safety and security properties.

Researchers have studied the possibility of automatically predicting users activity [151, 191]. Rashidi et al. [151] proposed Apriori algorithm to detect activity patterns in a smart home environment to automate users activities. Similarly, [191] used Bayesian networks to learn and predict users activities which can be used to generate personalised activities. These works focused on predicting and automating users activities which can be represented as workflows to be used as an input in our work (Chapter 4 and 5).

User activities are represented as workflows that are tied to a set of concrete underlying devices [65, 178, 142, 121]. This representation implies that an activity can only be fulfilled by exactly the same collection of devices specified in the workflow [142, 121].

Policy Generation

Existing works using offline/pre-defined policies such as MUD [101] are not sufficient for dynamic IoT environments [72]. The MUDs are defined offline by IoT vendors to specify what network access is required for a particular IoT device to work properly [101]. However, pre-defined policies are generally not fine-grained enough, especially for local connections, as they depend on how users are using their IoT devices to fulfil their activities. The best that a MUD can do is to restrict access between IoT devices by a specific manufacturer, which is coarse-grained access.

One approach to automatic policy generation is to capture all types of benign traffic in relation to any targeted offline IoT devices and then parse the captured traffic to generate a policy to be enforced in the deployment [73, 172]. Although this approach can generate fine-grained policies, it requires the training process to cover all possible communications a device can do to determine precisely all allowed local endpoints. This is infeasible for many IoT devices, as it depends on how they are used at the application level.

Some research has been done to automatically generate more fine-grained IoT policies according to devices that are being used to support users' new and ongoing activities. An example of such a method is proposed by Tian et al. [181] where access control is driven from the semantic of IoT apps source code, code annotation, and capability request. This work focuses on an app that controls an individual IoT device intending to prevent it from a behaviour different than it should. However, similar to the MUD, it mainly focuses on individual device behaviour and does not consider when multiple devices are used together to fulfil user activities.

In view of this problem, Al-Shaboti et al. [4] developed a new technique for fulfilling user activities using a user preference model to automatically generate IoT policy for these activities.

2.2.3 Multi-User IoT Sharing

The related work on IoT sharing and sharing policy language are presented below.

IoT sharing

Recent research has explored how users share their IoT devices [198, 58, 62] and suggests that sharing is influenced by trust between users, content accessed on the shared device, and accountability [58]. For example, guests may not be allowed to change the device configuration or use a Smart Speaker for purchasing goods, however, they may use it to play music.

Grag et al. [58] conducted a dairy study to show the preferences and constraints of sharing IoT devices. Users tend to share IoT devices with their family members as well as friends, co-workers, and guest. There are some policies reported in this work on how users coordinate their use of shared devices. Owners define how they want to share their devices, and they agree with the other users on these based on trust. They define with whom they want to share their devices (e.g. family members, guest users), and how they should use the shared devices. For example, owners may allow Alexa to be shared with their kids for homework tasks but not to be used for purchasing function. For privacy issues, users do not share their IoT devices for any functions that require access to personal data such as calendar and email if they are sharing the same account [58]. Others create separated accounts for each user if it is supported by the device to protect their personal data [62].

Jang et al. [90] have studied the problems that arise when multi-users use the same IoT devices. This research suggests that users expect devices to be accessed based on the functions they need to fulfil. Also, it suggests that the primary user (e.g. owner) needs to be able to control the privileges of other users. Authors in [62] studied the interactions between users and devices in a smart home. They observed that the person who installs IoT devices has an outsized role over other users.

These works help us understand users sharing mental models and guide our design for the sharing policy language.

Policy language

Most of the existing policy languages are technical and designed for network administrators and not for smart space users [99, 128]. Lara et al. [99] proposed a human-readable network policy that allows users to define alerts with network service and determine what security reaction should be taken. These policies require an expert to write a policy (what flow, what service, what reaction) and do not meet the requirements of ordinary smart space users.

There are a few special-purpose policies that are developed for smart space users. Yahyazadeh et al. [193] proposed EXPAT a system that allows users to use a policy language to express their expectations on how their IoT should work. Authors develop a policy language to enable users to express their expectations to avoid any physical breach. Users expectations focus on the outcome of the ongoing user activities automaton and not on expressing their sharing condition. Motivated by this work, we develop our sharing policy language for the purpose of expressing sharing decisions (presented in Chapter 5).

Authors in [198, 160] implemented a flexible access control mechanism that enables users to enforce role-based and location-based access control. Similarly, Jang et al. [90] proposed to use a user profile mechanism to deal with multi-user IoT. They used a role-based access model where they defined three roles that are associated with three privilege levels. Although it uses roles to give different access levels (e.g. admin users only can configure the device), users who have access can use any device's functionality. It also does not consider cross-device communication.

Another traditional model is the Chinese Wall model [22] which prevents resources that belong to the same Conflict of Interest (CoI) class to be used together. Similarly, if a user sharing policy is to prevent a user from using his device another user's device, then both users' devices will be in the same CoI class. However, a multi-user shared environment is more dynamic, and with hundreds of devices, the Chinese wall is not efficient and flexible enough to manage it.

2.2.4 Summary

This section introduces the related work on the three contribution chapters. First, it looks to the related work on using user input to derive fine-grained access control decisions. Second, it reviews relevant research on user activity representation as workflow, user preference, and its relation to the IoT access control. Third, this section reviews the related research on multi-user IoT sharing and relevant policy languages.

Chapter 3

User-Centric Smart Space Access Control Framework

3.1 Introduction

Smart space IoT devices often have coarse-grained access to the network, which makes the network vulnerable to various threats that can exploit IoT vulnerabilities. The malicious/compromised IoT devices are a threat that can exploit their network access to infect/compromise other IoT devices. This threat can be mitigated by applying fine-grained network access control to only enable an IoT device to communicate with the required devices based on user usage. This problem can be tackled from a technical point through enforcing the principle of least privilege (see Section 2.1.5), explained in 2.1.5. Many system have been proposed to control devices access to the network [101, 73, 17, 172, 70]. Access control is often applied in the network layer as it can be enforced across all types of IoT devices.

The majority of these methods ignored user input and are not efficient in capturing inter-device interaction. In other words, those methods designed to limit each individual device access to what it is manufactured to do using policies such as MUD policy [100, 101] (see Section 2.1.5). It is not easy to enforce access control based on how these devices are actually used without considering user input. This chapter focuses on enabling fine-grained access control in smart pace by supporting user access control input to customise IoT policies. Furthermore, a malicious device can still be a threat to the devices it has access to, after applying user policy. It can launch different attacks against other IoT devices such as a MITM attack using ARP spoofing or login brute force attack. This chapter introduces two security services that can be integrated with the framework to mitigate these attacks. In particular, ARP server to mitigate ARP spoofing attack and IDS to detect active attacks such as login brute force. It is worth noting that rule conflict problem can occur; however, this problem is well investigated in the literature. Methods such as the one in [186] can be used to detect potential rule conflict before committing a new rule. A smart home scenario will be used in this chapter as a typical smart space to present the framework and its components.

3.1.1 Chapter Goals

Motivated by a) the limited user control to enforce access control on the smart space and b) the SDN features such as centralisation, programmability, and the fine-grained network access control ability, a new framework is developed to allow users to customise IoT pre-defined policies and integrating these policies with common network security services.

This chapter aims at addressing the following objectives:

1. Developing a framework that enables fine-grained access control by allowing users to customise IoT pre-defined policy in smart space using SDN. The framework supports the users to control what devices can join their network, and they can see a visual representation of the network topology and the access rules on each link between their devices. Moreover, the framework also incorporates existing security services such as IDS.

2. Developing ARP spoofing mitigation technique and integrating IDS

with SDN to mitigate ongoing attacks. A novel SDN-based approach to mitigate malicious IPv4 ARP spoofing will be presented using an ARP server as a security service. ARP spoofing results in numerous attacks, of which the most noteworthy one is the Man-in-the-Middle (MITM) attack, host impersonation and DoS attacks. A trusted ARP server to all ARP requests using a pre-configured dataset of IP and MAC entries. Previous works proposed ARP mitigation as a control plane application which leverages port ACLs to permit hosts to send ARP packets with its IP/MAC combination [130, 37]. However, it cannot be applied to situations where more than one IP/MAC are associated with the same port (e.g. WiFi, virtual machines environment). Moreover, results in [130] show that ARP requests arrival time to the controller goes up to 14 seconds when ARP flooding attack is in progress which may cause DoS against the controller. The advantages of the ARP server are: (1) Secure ARP operations; (2) Eliminate the ARP broadcast messages; (3) Ease the legitimate ARP spoofing (e.g. ARP proxy) by configuring the ARP server. We show how the packet processing delay problem can be mitigated using high-speed packet processing technology. Moreover, automatic attack mitigation service is developed using IDS and SDN where the IDS is used to inspect network traffic and sends security alert to the framework which uses SDN to apply access control on the suspicious connections.

3. Demonstrating the feasibility of the proposed framework and its flexibility of supporting security services such as the ARP server and IDS through smart home IoT scenarios. Validating and evaluating the performance of the ARP server using Python *Scapy* library and Data Plane Development Kit (DPDK) implementations. Also, validating the IDS integration with the framework using a testbed scenario.

3.1.2 Chapter Organisation

The remainder of this chapter is organised as follows. Section 3.2 presents the IoT fine-grained access control framework, pre-defined IoT policies, and user policy use cases. Two security services (i.e. IDS and ARP server) are described in Section 3.3. The framework prototype implementation and validation is presented in Section 3.4. The same section also presents security services evaluation and validation. Section 3.5 concludes this chapter.

3.2 Access Control Framework

3.2.1 The Framework Requirements

The framework imposes some requirements in order to achieve its goals as follows:

- **Pre-defined access policy.** Apply pre-defined IoT device policies, such as MUD, as base-line policies that can be overwritten later by users access rules.

- **User-defined access control.** Provide users with the ability to define and enforce network access rules that facilitate internally (within the network) and external (to/from the Internet). User rules should overwrite the pre-defined policies. To enable users to effectively exercise their control the framework should provide the following functions: (a) Lists the joined devices and the associated internal and external access policy (display the current network state); (b) Allows users to add/remove existing access rules for any device in the network; (c) Provide a visual map for the network topology and the network policy on each link in the network to convey the network policies to users in a simple way.

- **Security services integration.** The framework has to be able to integrate with local and remote security services. Such that it can examine the relevant network traffic and provide the corresponding security decision. This requirement will allow the user to delegate different security functions to different security providers, unlike previous solutions where all security functions are delegated to one provider (e.g. ISP).

In order to support these requirements, SDN technology has been chosen as a key technology to implement the framework due to its programmability feature. SDN controller can easily enforce Network Access Control (NAC) by pushing OpenFlow rules into the data plane in a dynamic way. Also, Network Function Virtualisation (NFV) for security services.

3.2.2 Framework Design

The framework is shown in Figure 3.1 which includes both local and remote components as follows:

1. **Data plane** is the actual hardware switch that all IoT devices are communicating through it.

2. **SDN controller** is the main component responsible for enforcing network access control on the data plane through OpenFlow protocol, see Section 2.1.3 for more details about SDN architecture. The SDN controller is utilised by the framework security agent to enforce access control.

3. **Security agent** is an interface between the SDN controller at one side and the other security services from the other side (i.e. user control panel, IoT policy manager, and local/remote security services). It plays an important role to fulfil the security integration requirement. Security agent receives the access control rules from the user control

panel, IoT policy manager, and the security services, then it configures the SDN controller to enforce these rules in the data plane. It also controls packet forwarding/mirroring to local and remote security services.

4. **User control panel** is used to get user access rules and pass them to the security agent. It shows the current status of the network policy, the devices in the network and allows users to customise these policies. The user manually enters rules similar to one that are used in a home router firewall.

5. **Remote dispatcher** is a local component that is used to process all mirrored packets and forward them to the corresponding remote security service based on the security agent instructions (i.e. dispatcher control messages).

6. **Security services** are local and remote systems that provide security functions. A security service examines network traffic and generates security alerts back to the security agent. It can be a local security service such as IDS, or it can be a remote security service such as secure DNS and Secure Web Gateway (SWG).

3.2.3 Framework Design Decisions

The rationale behind the framework components design is as follows.

The security agent and dispatcher have a different responsibility, and single responsibility design principle is applied to separate them. The security agent function is to enforce the network policy through the northbound controller API as a response to the received alerts, and the input from users access rules and IoT policy manager. A low-cost controller can handle up to 10,000 new flow per second, as stated in Ethan experiment [24]. However, for applications that require processing per packet, a good option is to forward such packets to a specialised node (i.e. NFV) and not

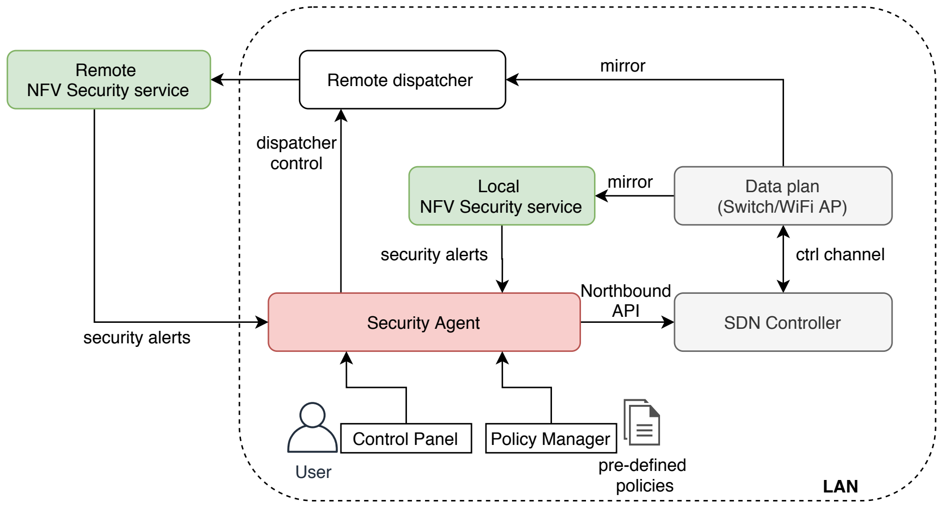


Figure 3.1: The proposed framework

overload the controller. Therefore, NFV dispatcher works by forwarding the traffic based on the security agent dispatch control to security services. The dispatcher node forwards selected traffic for further inspection. For example, the security agent mirrors the traffic to the dispatcher to be forward to a DNS inspection security service. If a malicious DNS traffic is detected, the security service sends alert to the security agent, which then installs appropriate rules to block or quarantine the suspicious hosts.

Separating the security agent from the controller has several advantages: (a) it keeps the controller simple to perform its primary task (controls the data plane) and not interfere with direct interaction with inputs from users, IoT manager, and security services; (b) Security agent becomes independent from the controller; hence, any SDN controller can be used; (c) The security agent and the controller can be updated or upgrade independently without interfering each other.

Due to the scalability problem in the SDN control plane, the dispatcher is designed as NFV. The dispatcher needs to parse mirrored network traffic and forward them to the security services. Developing such application in the control plane may exhaust the control plane resources (i.e. control channel bandwidth, controller CPU and memory) since all requested network traffic have to be passed to the controller through the control channel (*PACKET_IN*), [79].

3.3 Network Security Services

To mitigate threats that are not captured by the security policies, two security services have been developed, namely, ARP server and IDS.

3.3.1 IPv4 ARP Server

IPv4 ARP server has been proposed as a security service to mitigate ARP spoofing attack. It also shows how security services can be integrated into the proposed framework. ARP spoofing is a common way to launch a MITM attack. This motivates researchers to propose different solutions. A recent study shows that two of the major OSs (i.e. Windows and Apple MAC) are vulnerable to the ARP poisoning [184]. That means IoT that runs an embedded version of these operating systems are even more vulnerable.

There are at least two options to mitigate IPv4 ARP spoofing using SDN. One option is to build an application on the control plane to enforce flow rules at the switch ports to drop spoofed ARP reply. However, more than one host may be connected to the same port (e.g. WiFi). Moreover, the control plane is not an ideal place to implement data plane processing functions [79]. Alternatively, the ARP server can be designed as an NFV security service node. The advantages of using this method are that a trusted ARP server can provide three main functions:

1. Secure the ARP operations by providing ARP replies through a trusted entity. For example, in case where there are malicious hosts on the network, if a forged ARP reply is sent, it will be dropped by the

switch. This is because all ARP replies are only allowed to come from the ARP server which mitigates malicious ARP spoofing.

2. Eliminate the ARP broadcast messages, as all ARP requests will be forwarded to the ARP server only. For example, in a network with tens or hundreds of hosts, ARP requests will be forwarded to the ARP server only, rather than being broadcasted to all hosts; hence, save the bandwidth and mitigate against passive discovery where a malicious host can discover existing devices through their ARP requests.

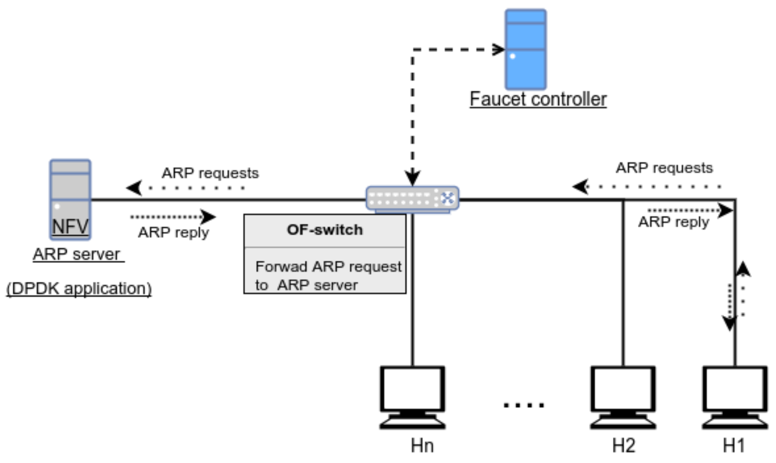
3. Ease the legitimate ARP spoofing (e.g. ARP proxy) as it can be implemented in the ARP server. For example, in some cases gratuitous ARP requests are required for setting a backup server to take over for a defective server and transparently offer redundancy. This can be done in a controlled and simple manner using the trusted ARP server.

The ARP server can utilise the DHCP leases file to build the ARP table. We assume if any static IPs are required, then they are reserved through the DHCP server as such they are included in the leases file.

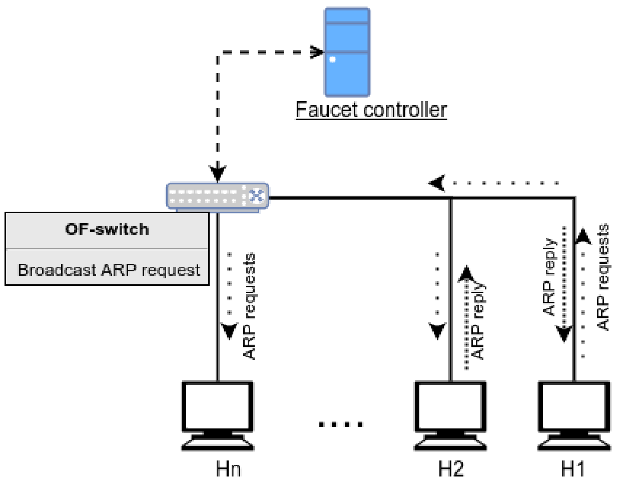
3.3.2 Intrusion Detection System (IDS)

IDS is used as a use case of a dynamic local security service that inspects network traffic and sends security alert to the security agent on any suspicious connection. The goal is to mirror IoT network traffic to an IDS, which inspect the traffic (e.g. looking for password guessing attack) then alert the security agent. The security agent receives the security alert and uses the SDN controller to block the connection or/and quarantine the compromised host.

Figure 3.3 shows how the framework can incorporate IDS as an NFV. For example, if a malicious host sends a malicious traffic to another host



(a) ARP operation with ARP server



(b) Conventional ARP operation

Figure 3.2: ARP operation with/without ARP server

in the network, the switch will mirror this packets to the IDS. Although the first few packets will reach the destination host, the IDS will notify the Security Agent which then installs deny rules in the SDN controller. Finally the SDN controller will install OpenFlow rules in the switch to block the malicious traffic.

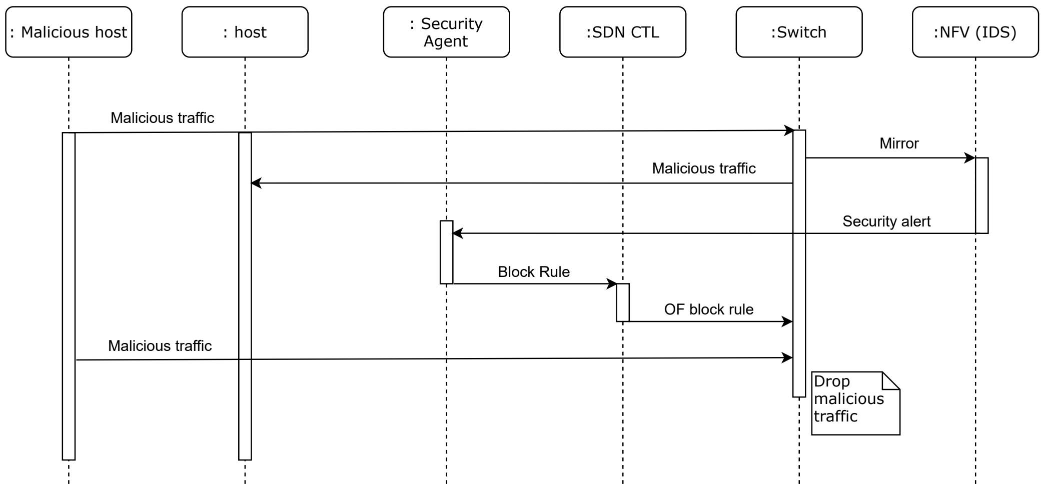


Figure 3.3: IDS integration into the framework for automatic access control.

3.4 Evaluation

In this section, a smart home scenario is built as a typical smart space to validate and demonstrate a prototype of the proposed framework. Three common attack activities have been used to validate the framework: network scanning, ARP spoofing, and malware file transferring.

We start by presenting the experimental setup. Then three experiments are conducted to (1) validate the network fine-grained access control using three scenarios; without any access control, with IoT device policies, and finally with user access rules; (2) validate the security services integration with the framework; (3) Evaluate the performance of a typical smart space OpenFlow-enabled switch.

3.4.1 Experimental Setup

Figure 3.4 illustrates the experimental setup of a smart home which includes the following:

- An OpenFlow-enabled switch (TP-link) runs Open vSwitch (OVS).

The switch supports WiFi and Ethernet connection and connects the smart home network to the Internet.

- A gateway and DHCP server runs in a raspberry Pi3.
- An Android smartphone that is used by the user to access the network control panel.
- A laptop runs Windows 10 operating system represents a general-purpose device.
- Ubuntu server runs the security agent and the web server that hosts the user control panel.
- A Kali-Linux Virtual Machine (VM) that imitates a malicious IoT device (it will be considered as a malicious IP-camera). The IoT malicious activities do not rely on any VM capabilities and were only implemented using virtual machines due to limited availability of resources.
- An open-source SDN controller (Faucet) is running in a raspberry Pi3 and control the OpenFlow switch via an Ethernet link.

The hosts are shown with the associated services running on each to illustrate later what services are accessible by the malicious IP Camera under in each experiment.

The framework is implemented using Python and hosted locally in the Ubuntu machine including the security agent, a simple version of IoT usage policy manager, user control panel, and the ARP server. The IoT usage policy manager retrieves MUD files from a local directory, compile it into Faucet ACL and sends it the security agent to be applied. The user control panel is implemented as a web application that is accessible through user Android mobile and it offers four functions:

1. Display the joined devices attributes (i.e. name, IP, MAC).

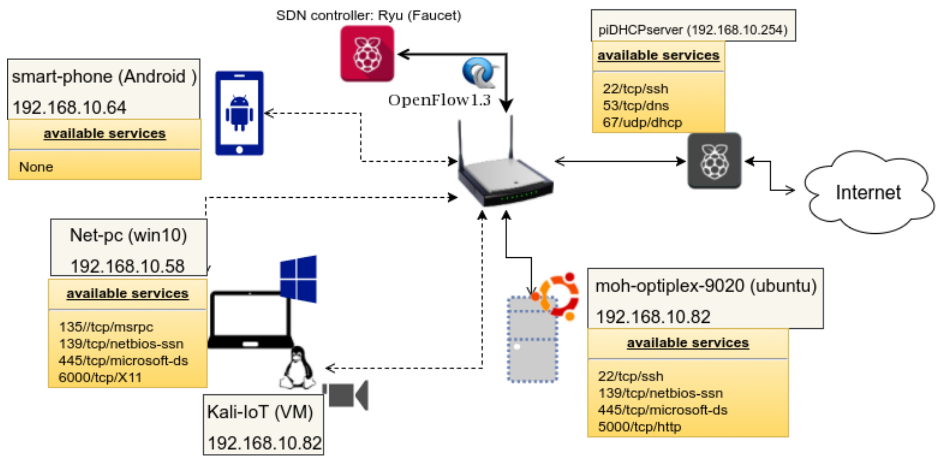


Figure 3.4: The smart home setup

2. Lists the blocked devices: those devices that are trying to connect but not permitted by the user to join the network. Figure 3.5 shows the main page users presented to after they log into the system.

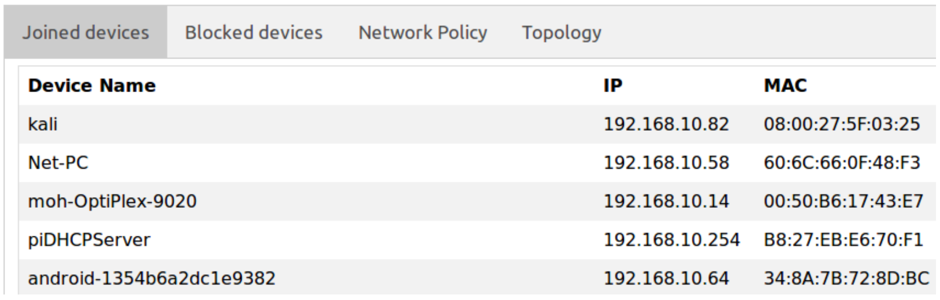
3. Lists the current network access policy here the user can add access control policy for any joined device.

4. Visualise the topology of the current network connection along with the policy applied for each link. For example, Figure 3.6 shows the devices connected in the network, the links between them, and for each link it shows the protocols allowed between the corresponding pair of devices.

The ARP server is hosted on a virtual machine inside the Ubuntu machine and has a dedicated network connection to the switch.

3.4.2 Network Access Control Validation

The risk associated with no access control, IoT policy only, and IoT policy with user access rules are all demonstrated as follows:



Device Name	IP	MAC
kali	192.168.10.82	08:00:27:5F:03:25
Net-PC	192.168.10.58	60:6C:66:0F:48:F3
moh-OptiPlex-9020	192.168.10.14	00:50:B6:17:43:E7
piDHCPServer	192.168.10.254	B8:27:EB:E6:70:F1
android-1354b6a2dc1e9382	192.168.10.64	34:8A:7B:72:8D:BC

Figure 3.5: Main user control panel: Joined devices

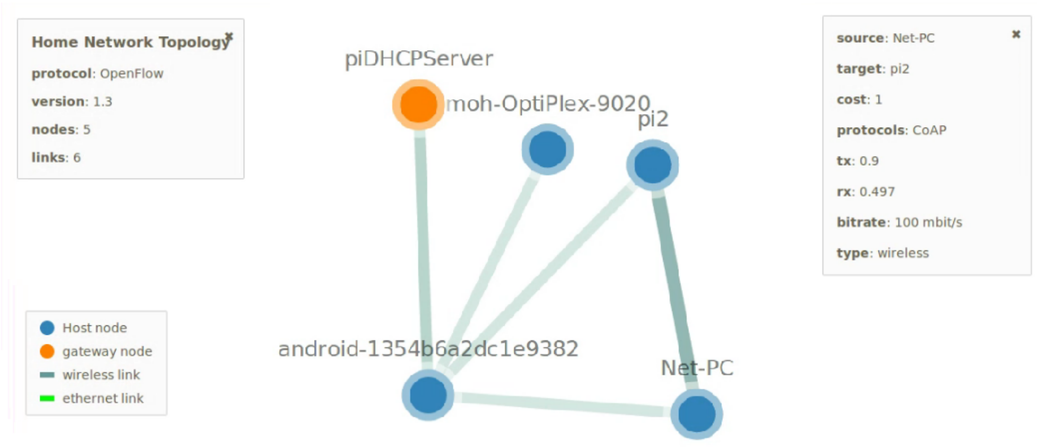


Figure 3.6: Visualised access policy.

Without Access control

In this test, the switch has been running without any access control between the devices inside the LAN, which means IoT devices joined the network with full access to any other device in the network and vice versa. To demonstrate the risk associated with such settings LAN and WAN scan have been performed using a malicious IoT device (i.e. Kali VM) to discover the available services. The Kali Linux imitates a malicious IP camera and uses *nmap* command *(nmap -sS -Pn 192.168.10.14,58,64,254)* to scan the top 1000 TCP ports on the LAN network [110]. Nmap options are *-sS* for SYN scan and *-Pn* to scan without pinging. Scanning results are summarised in Table 3.1, not surprisingly all TCP services on the networks were accessible, see Figure 3.4 to find the available network services in each host. The same scanning experiment is repeated on an Internet host (i.e. *hackthissite.org* is a public host deliberately vulnerable and used for educational purposes). Table 3.2 shows the scanning results where the IoT can access the host through all the available network services. From the two tests (i.e. LAN and WAN scan) it is clear that a compromised IoT device can be used for malicious activities, if compromised, either against the local network or the Internet hosts.

Table 3.1: TCP port LAN scan results

Target	Services
(Optiplex 9020)	22/ssh, 139, 445, 5000/(HTTP)
Net-PC	135, 139,445, 6000
Android	None
piDHCPServer	22/ssh, 53/domain

With Pre-defined IoT Policy

In this experiment, the IP camera MUD policy is generated by online utility [100] to validate enforcement and discuss security improvement.

The MUD access policy allows local connection to and from the camera and restricts its external connections to only *www.hackthissite.org* (imitates the Camera cloud service) using TCP port 443. The MUD file is imported through the IoT usage policy manager, which then gets compiled into OpenFlow rules and installed by the security agent in the switch through the Faucet controller.

To validate the functionality of the IoT policy, we repeated the scan in the previous experiment again (i.e. *nmap* scanning). When *ip-cam.json* MUD file is applied, *nmap* reports that ports 22 and 80 are filtered which means they are blocked by an intermediate entity (i.e. the switch), and only port 443 is accessible, as shown in Table 3.2. Therefore, MUD reduces the potential threat of the IoT to the Internet by limiting its Internet access to the least required access. With respect to the LAN connections, MUD allows full access to the IP-camera from any device (e.g. laptops, tablets, smart-phones), since this depends on which devices users are using with the camera.

Table 3.2: TCP port WAN scan results

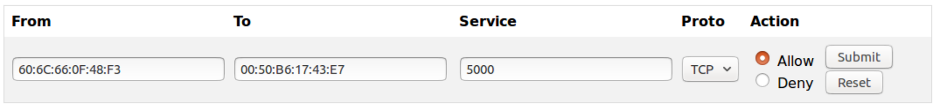
Target	Services	MUD
hackthissite.org	22/ssh, 443/HTTPS, 80/HTTP	None
hackthissite.org	(22, 80) filtered, 443/HTTPS open	ip-cam.json

With User Access Rules Input

As we discuss above, the IP-camera MUD allows any available device in the network to access it. As the manufacturer cannot specify if a mobile/pc/laptop is allowed to connect or not, user access rules will be applied here to show how it reduced LAN attack surface. Users may want to restrict access to their IP camera to only a specific host.

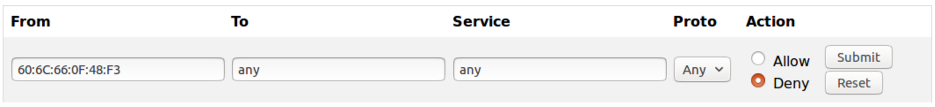
In the proposed framework, users can leverage the control panel and check the current access policy for the IP-camera and modify the access permissions as required. For example, in Figure 3.7a users allow TCP connection on port 5000 from their desktop (Optiplex 9020) to the IP-camera (i.e Kali), and block any other connections as shown in Figure 3.7b. A final access policy is displayed in Figure 3.7c. The scan is repeated on the network to verify that the system enforces user's rules, and how they reduce the attack surface. Scan results in Table 3.3 show that only port 5000 is accessible and all other network services are blocked.

Video footage for the prototype demo that shows the topology and access control visualisation is available here [2].



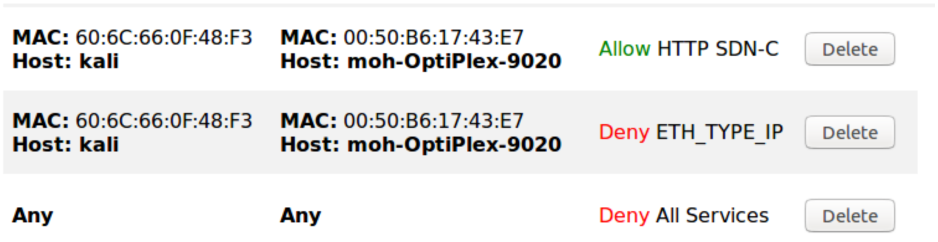
From	To	Service	Proto	Action
60:6C:66:0F:48:F3	00:50:B6:17:43:E7	5000	TCP	[✓ Allow] Deny

(a) User allows access to the IP-camera TCP port 5000



From	To	Service	Proto	Action
60:6C:66:0F:48:F3	any	any	Any	Allow [✓ Deny]

(b) User blocks any access to the IP-camera



MAC: 60:6C:66:0F:48:F3 **Host: kali**	**MAC:** 00:50:B6:17:43:E7 **Host: moh-OptiPlex-9020**	Allow HTTP SDN-C	Delete
MAC: 60:6C:66:0F:48:F3 **Host: kali**	**MAC:** 00:50:B6:17:43:E7 **Host: moh-OptiPlex-9020**	Deny ETH_TYPE_IP	Delete
Any	**Any**	Deny All Services	Delete

(c) Access list view for IP-camera

Figure 3.7: User control panel: DAC enforcement

3.4.3 Security Services

The two security services are tested; the proposed ARP server performance is evaluated against the conventional ARP broadcasting operation, and the Zeek IDS integration with the framework is validated.

Table 3.3: TCP port LAN scan after DAC

Target	Status	Services
(Optiplex 9020)	Up	5000/(HTTP)
Net-PC	Up	None
Android	Up	None
piDHCPServer	Up	None

ARP Server Evaluation

This test measures the performance of the proposed ARP server 3.3.1 under different load compare to the normal ARP operation. The ARP server initially implemented using Python3 *Scapy* library. However, it doesn't scale as its performance declines as the number of hosts increases. Then ARP server is implemented using the Data Plane Development Kit (DPDK) [149]. DPDK is a network acceleration technology that contains a set of data plane libraries and network drivers. It allows an application to process packets directly and bypass the kernel; hence, it reduces the packet processing time.

SDN controller (i.e. Faucet) installs OpenFlow rules into the switch to forward all ARP packets coming from any host to the ARP server port. These OpenFlow rules are installed only once during the setup and will forward all ARP requests to the ARP server physical port. For example, as shown in Figure 3.8, if host H1 sends ARP request asking for host H2's MAC, its request will be forwarded to the ARP server instead of being broadcast to all hosts. Then ARP server works by consulting its ARP Table and form an ARP reply packet with its own MAC address in the Ethernet frame sender field and with the corresponding MAC and IP in the ARP sender header.

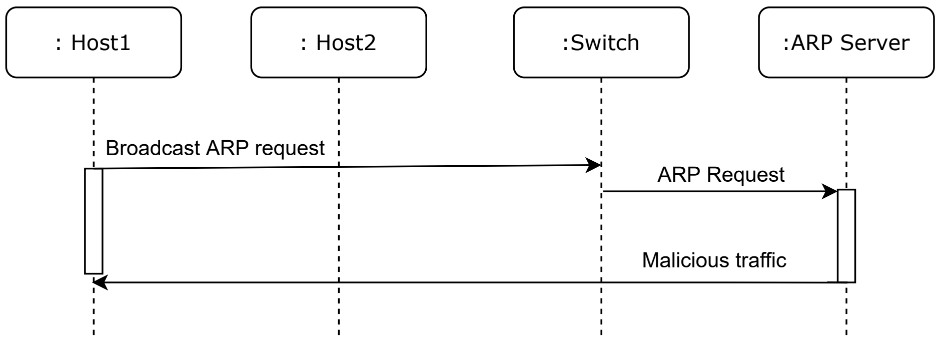


Figure 3.8: ARP server operation.

The experiment was conducted using Mininet, with two setups similar to Figure 3.2; one without ARP server (i.e. conventional ARP operation) and the other with ARP server. For each setup, the number of hosts starts with ten hosts and increases to 50 hosts. Linux *Arping* utility is used to generate parallel ARP requests to measure if ARP server can scale. Arp request is created such that each host is arping the subsequent host (i.e. h1 is arping h2, h2 is arping h3, and so on).

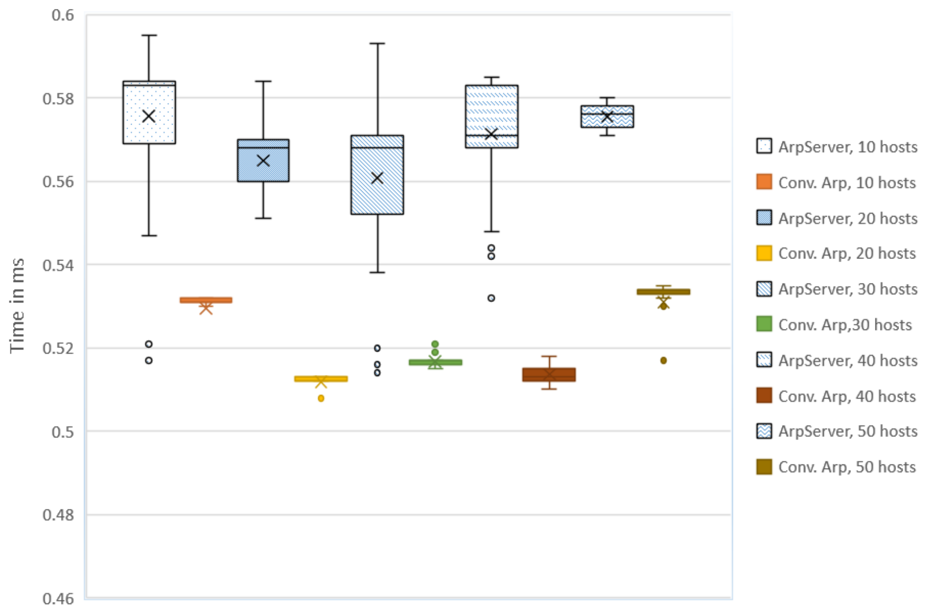


Figure 3.9: ARP response time (DPDK ARP server vs conv. ARP operation)

Results in Figure 3.9 show that the ARP response time using DPDK ARP server is slightly higher compared to the conventional ARP by 50μs. For both settings (e.g. conventional ARP and ARP server) there is no significant impact of an ARP load up to 50 parallel ARP requests, as all ARP responses are between 0.5ms and 0.6ms. However, results were different from the Python ARP server implementation. Figure 3.10 shows that the ARP response time of the Scapy ARP server is significantly higher (70ms and more) than compared to the conventional ARP and DPDK ARP server (less than 0.6ms when runs on a single core @3.60GHz). DPDK ARP server outperforms the centralised ARP server in [31] in which the ARP server was implemented as an application in the control layer and gave response time of 80ms for 25 hosts.

Scapy ARP server is also highly affected by the number of parallel ARP requests, as it starts to drop ARP packets when there are 30 parallel ARP or more (see Figure 3.11). ARP server also tested against ARP spoofing attack and experiment shows that all fake ARP packets sent by attacker's host have been dropped the switch as only ARP response accepted from the ARP server physical port.

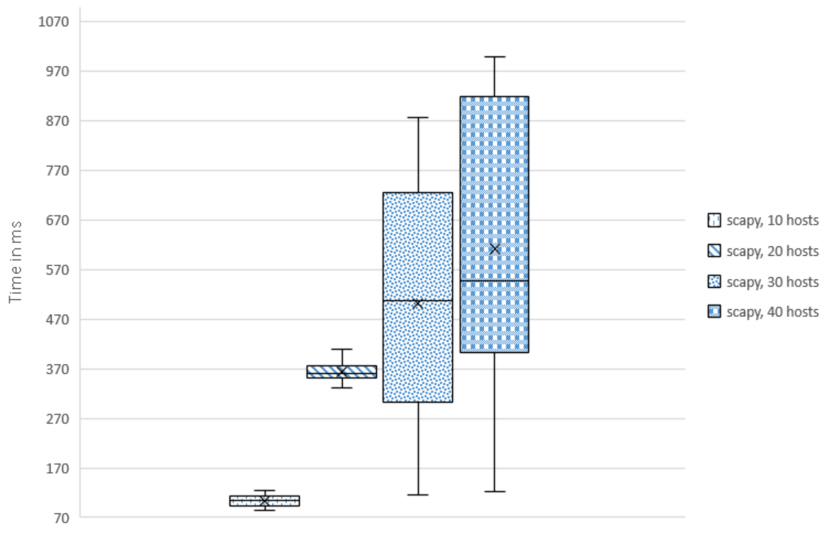


Figure 3.10: ARP response time (Scapy ARP server)

IDS Integration Validation

Unlike the ARP server, IDS is a security service that is integrated with the proposed framework to enforce access control rules dynamically. The testbed implemented using Docker containers to run Zeek IDS, Faucet SDN controller, and Python script (represents the security agent of the framework) as shown in Figure 3.13. Two more containers are used to represent two IoT devices where one is compromised and trying to spread malware using HTTP protocol in the network. Faucet is configured to mir-

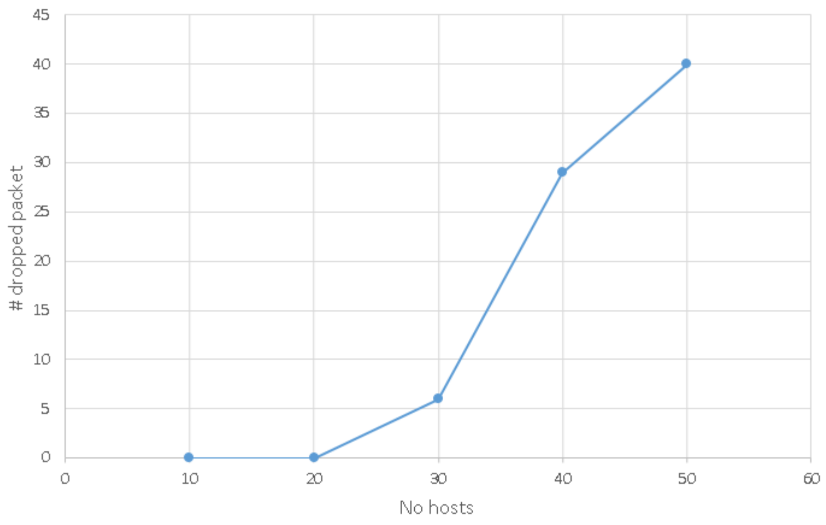


Figure 3.11: Scapy ARP server performance with the increase of number of hosts

ror all traffic to Zeek IDS. Zeek is configured to check for any malicious file with a specific hash value using the Zeek script in Figure 3.12.

For sake of illustration, an HTTP request is sent from *host2* to fetch the malicious file from *host1*. The HTTP response carrying the malicious file is captured by Zeek and triggers its hash file event that forward a malicious file alarm to the security agent script, see Figure 3.12. The security agent successfully processed the alert and updated the Faucet ACLs by adding a new rule that drops any new connections from the compromised host. The source code of this testbed is available in the GitHub repository [3].

3.4.4 OpenFlow-enabled Switch Performance

The goal of this test is to quantify the bandwidth of the peer-to-peer Ethernet/WiFi connections in the LAN network and to study the effect of the NAC on the network performance. A commonly used utility (i.e. *iperf*) is utilised to measure the TCP and UDP performance between two Raspberry Pi3 (model B v1) that represent two IoT devices (e.g. Camera stream-

```
event file_sniff(f: fa_file, meta: fa_metadata)
{
 if ( ! meta?$mime_type ) return;
 if ( meta$mime_type == "application/x-executable" )
    Files::add_analyzer(f, Files::ANALYZER_MD5);
}

event file_hash(f: fa_file,
                kind: string, hash: string)
{
 if (kind== "md5"
    && hash == "8e5b325156981e0bcba714dc32f718c5" ){
    print "Bash binary file md5!";
    for ( cid in f$conns )
     {
       drop_connection(cid, 3600 secs);
     }
 }
}
```

Figure 3.12: Malicious malware event trigger.

ing to a smart TV).

Two setups are used; one for WiFi link and one for Ethernet link. The two Pis are connected to the switch through WiFi and in another setup via Ethernet ports. In each case, the bandwidth of the link is measured under no access control rules enforced, and when the access is restricted to the TCP/UDP *iperf* port. As Table 3.4 shows network access rules have no significant effects on the switch neither in the Ethernet connection nor in the WiFi links.

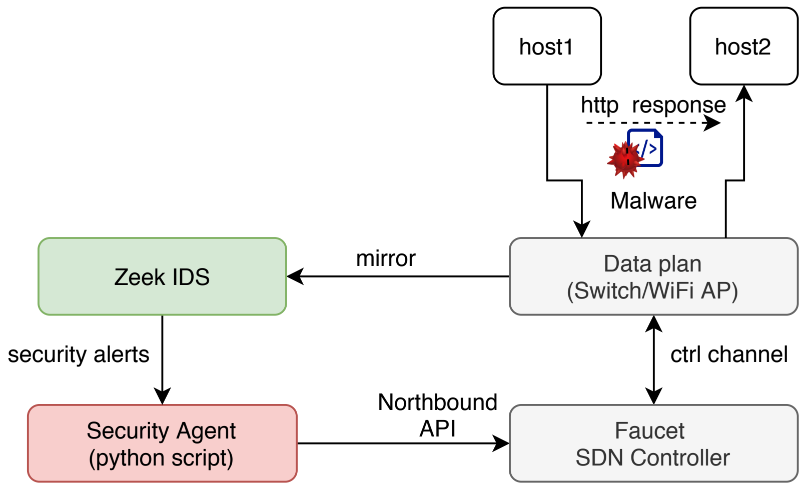


Figure 3.13: Zeek IDS and Faucet SDN controller integration for automatic access control.

3.4.5 Evaluation Summary

Deploying IoT without access control policy in place poses high-security threats to the internal and external network. Enforcing MUD policy can reduce the external and internal attack surface. However, it has coarse-grained internal access which can be customised using user access control. IPv4 ARP server validation shows that it protects against ARP spoofing and its DPDK implementation performance was sufficient for the smart

Table 3.4: Bandwidth With/without NAC

NAC	No NAC		Restricted NAC	
Conn. type	TCP	UDP	TCP	UDP
Eth to Eth	94.1 MB	1.05 MB	94.1 MB	1.05 MB
WiFi to WiFi	21 MB	1.05 MB	20.8 MB	1.05 MB

home network. A packet processing services such as ARP server need a special processing driver to reach the packet faster such as DPDK. Network access rules have no performance effect on a typical OpenFlow-enabled switch.

3.5 Conclusion

In smart spaces, users play an important role in protecting their devices. This chapter presents a new framework that incorporates users' access rules to customise pre-defined IoT policies (e.g. MUD policies) to enable fine-grained access control. Furthermore, the framework also integrates network security services. Two network security services have been presented as examples of network security services. An IPv4 ARP server is presented as a security service that works within the framework to mitigate ARP spoofing attacks. In addition, IDS service is used to show the feasibility of integrating existing security solutions into the proposed framework.

The proposed framework is evaluated using a smart home scenario and results show its usefulness in reducing the attack surface. Furthermore, the results indicate that the DPDK ARP server was able to manage up to 50 parallel ARP requests with an overhead of 50 μs compared to the conventional ARP response time. It outperforms other ARP server implementation in the literature. The IDS integration was validated and the framework was able to dynamically respond to IDS alerts by quarantining the malicious host in the network.

Chapter 4

Automatic Activity Fulfilment and Fine-grained Policy Generation

4.1 Introduction

The emerging Internet of Things (IoT) has led to a dramatic increase in type, quantity, and the number of functions that can be offered in a smart environment. Future smart environments will be even richer in terms of number of devices and functionality provided by them. With the increasing number of consumer IoT devices, managing access control of these devices becomes very challenging. This poses two major challenges: (1) Users have to search through a vast number of IoT devices to identify the suitable devices that satisfy their preferences; (2) It is extremely difficult for users to manually define fine-grained security policies to support workflows involving multiple functions.

User preference can capture their choices related to security, privacy or even perceptions of quality. For example, users may prefer to use brand X devices for a sensitive function such as audio recording due to the manufacturer security reputation and data privacy policy [23, 75]. Therefore, user preference must be considered when selecting which devices should be used to fulfil their activities. Various information about a device can be used to build a user preference model such as device brand and location.

Users are not always interested in manually specifying the devices to be used to fulfil their activities. In a dense smart space, there could be many alternative devices that can be used to accomplish a specific activity function. This makes the task of creating a new activity a challenge for users, as they have to select which device to use for which function. Users may not be interested in that; rather, they are interested in the actual activity function.

The proposed techniques in this chapter use the concept of workflow, which is an abstract representation of user activities as a set of required functions to be fulfilled without specifying what devices should be used. The concept of workflow abstraction based on devices' functions is depicted in Figure 4.1. For example, instead of a user building a concrete activity by selecting the devices, Brand_X alarm triggers Brand_Y coffee maker, a workflow can be defined in an abstract way as alarm function triggers coffee maker function, see 4.1. At the deployment, our proposed system can automatically select the suitable underlying devices (e.g. Brand_X alarm and Brand_Y coffee maker) using user preference. Hence, users can focus on the functions of the activity rather than devices selection which can be a tedious task in a large IoT network and doesn't support dynamic environment.

Supported by a user preference model, our proposed approach automatically selects the suitable collection of devices to fulfil users activities. There are many technologies in the literature to accurately learn preference models [148, 54]. From which, there are two main approaches to modelling preference: the value function approach in which an abstract utility value is assigned to each alternative under consideration; and the preference relations which captures the relation between alternatives options and present them as ranking, partial order relation [148]. The later technologies are considered to build a preference model that can capture the relation between devices when they are selected to fulfil users activi-

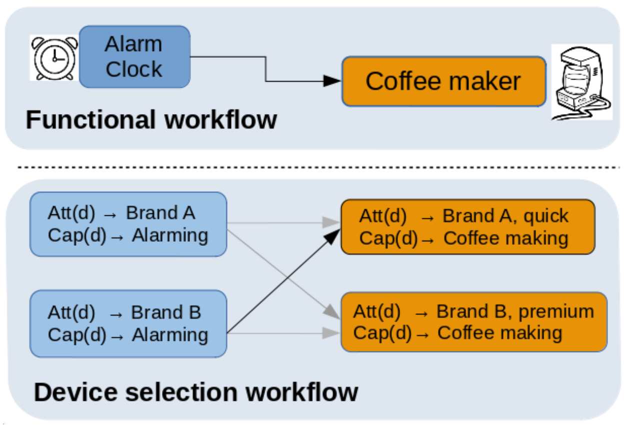


Figure 4.1: Functional workflow abstraction

ties.

4.1.1 Chapter Goals

This chapter aims at addressing the following objectives:

- Designing an appropriate activity representation that can be used to enable activity and access control automation.
- Modelling the smart space and user preference to enable activity fulfilment and access control automation.
- Formulating smart space automation as a mathematical optimisation problem.
- Studying the performance of search algorithms to automatically select devices to fulfil the activities while maximising user preference.
- Developing a simple method to systematically generate fine-grained network access policies to support user activities in a secure manner.

Note that the users only need to define their activities as functional workflows. Then the proposed methods in this chapter do the following

a) automatically select the suitable devices to fulfil the workflows function requirements and then systematically generate fine-grained network access policies to support user activities.

4.1.2 Chapter Organisation

This chapter includes two sections. Section 4.2 studies how to automatically fulfil user activities given a user preference model as an input. Section 4.3 studies how to automatically generate network and application access policy to support users activities considering the least privilege principle.

4.2 Automatic Activity Fulfilment

Existing research has considered the problem of how to automatically predict users activities using a set of sensors [151, 191]. For example, in previous research, the key idea is to analyse user activities, such as user movement in the house, to predict what activities user may need next. This work aims to propose a complementary approach that uses a functional workflow to represent user activity as user input. A functional workflow can also be obtained intelligently by using the methods developed in [151, 191]. Then, the functional workflows will be used to find the preferable set of devices to satisfy workflow functions while considering user preference.

4.2.1 User Activity Representation

The proposed technique uses a functional workflow to represent user activity such that it captures the activity function requirements without specifying which devices should be used to fulfil them. Users are not always interested in manually specifying all the devices to be used for any activity.

They are more interested in abstractly describing the required functionalities, for example, in the form of a flow of functions. However, currently, a workflow is often tied to the underlying devices [65, 21, 178, 142, 179], such that a user cannot use any pre-defined workflows unless he/she has exactly the same collection of devices specified in the flow [142, 179]. However, there could be alternative devices that can be used to accomplish the same activity.

Decoupling workflow's functions from underlying devices enable users to share and re-use many existing flows. Decouple architecture successfully eases the management and control in many existing problems [55] such as network management (i.e. SDN [47]) and web service composition [74]. Previous research focus on the cost, execution time, management and our focus here is security. Existing work represents user activity using a flow-based programming [19]. A user activity is represented as a workflow of wired processes that is hardcoded to represent specific IoT devices. Hence, such activities cannot be re-used or shared as it is tied to specific devices. This representation is common and supported by many IoT frameworks such as NodeRed [142], Mozilla Gateway [121], and Microsoft flow [52]. On the other hand, the proposed workflow ties the activity to the functions required, rather than to the underlying devices that can do these functions, this concept is depicted in Figure 4.1. For example, instead of building a flow by selecting the devices, Brand_X alarm triggers Brand_Y coffee maker, flow can be defined in an abstract way as alarm triggers coffee maker, see 4.1. At the deployment, our proposed system can automatically select the suitable underlying devices (e.g. Brand_X alarm and Brand_Y coffee maker) based on the flow requirements and user preferences. Hence, users can focus on what the activity function rather than the device selection, which can be a tidies task in large IoT network.

4.2.2 Smart Space Modelling

The smart space is abstracted into two models: (1) network model that captures the IoT devices and their capabilities, and the corresponding network access requirements to fulfil each device function; (2) the user preference model which represents user preference of using a set of devices for a set of functions.

Network Model

The smart space network is modelled as a collection of devices represented by $N = \{d_i : i = 1, 2, 3, ..., n\}$. Each device d in the network is represented by a tuple of (1) a set of attributes $Att(d)$, (2) a set capabilities $Cap(d)$, and (3) a set of network requirements for a particular function f, given by $Net_{req}(d, f)$. The network model is presented in Equation (4.1).

$$\forall d \in N, d =< Att(d), Cap(d), Net_{req}(d, f) > \tag{4.1}$$

Each attribute $a \in Att(d)$ corresponds to a device-specific property that may derive user preference of using a specific device over the others. For example, a coffee maker can have several attributes such as *type, brand, quality of coffee produced, time of preparation*. On the other hand, the set of capabilities $Cap(d)$ of device d refers to the set of functions that can be executed by the device. A device can have multiple attributes and can support multiple capabilities. For example, in Figure 4.1 the first coffee maker has attributes of *Brand A* and can quickly prepare coffee (i.e. *speed*), while it has only one capability which is making coffee. The network requirements R for a device d to execute a function $f \in Cap(d)$ is given by $Net_{req}(d, f)$ as shown in Equation (4.2). Such that $r \in R$ is a tuple of required access to the network for a device d to execute a function $f \in F$, where F is the set of workflow functions. Network requirements Net_{req} function will be utilised for automatic policy generation, to be introduced in Section 4.3. The proposed system architecture is depicted in Figure 4.3.

$$R = Net_{req}(f, d)\, \forall d \in D_s, f \in F \tag{4.2}$$

Network requirements can be extracted based on several existing techniques such as capturing and analysing IoT traffic [73, 172]. The aim of the network requirements to identify what protocol a device use for what function. Then when a device is used to fulfil activity function, the corresponding protocol should be allowed. Network requirements can also be extracted from MUD, which clearly defines the type of protocols used by specific devices. However, MUD cannot specify precisely the endpoints that devices should have access to prior to deployment. In our approach, the endpoint defined when the devices selected to fulfil the activity. Hence; fine-grained access can be derived.

A user drives the operations in a network by creating one or more activities as functional workflows for the network to fulfil. A functional workflow W is represented as a Directed Acyclic Graph (DAG) $W =< F, E >$, where F is a set of functions, and E is the dependencies among them. For a workflow W to be feasible, each function $f \in F$ must be satisfied by at least one device in the network N as shown in Equation (4.3).

$$\forall f \in F,\, \exists\, d\, , f \in Cap(d) \tag{4.3}$$

Meanwhile, every edge $e \in E$ represents the dependencies between a pair of functions. For example, if an activity includes an IP camera that streams to a TV, then TV and camera should be able to communicate using the protocol of streaming. To support this dependency, the access policy should include a rule that allows the TV to connect to the camera to fulfil the workflow.

User Preference Model

In order to determine the suitable devices for an activity, a flexible *user preference model* $M(F, D)$ is adopted to quantify user preferences of using any devices for various functions. *User preference model* can answer questions such as how likely a user will prefer to use any subset of devices $D \subseteq N$ for given workflow functions F. Any preference models that can quantify user preference can be used in the proposed system. Such a user preference model can also be obtained by using a variety of machine learning techniques such as preference logic, fuzzy logic, neural networks [148].

In our experiments, Bayesian networks technique is used to represent a user preference model. Bayesian networks [39] is a common technique for modelling user preferences and predict users activities [191, 153, 155]. For example, the user preference model in Figure 4.2 can tell our system that a user prefers to use Brand_X alarm device and Brand_Y coffee maker, their joint probability is the highest among any other combination. Using the graphical structure of the Bayesian networks it is easy to calculate the probability for a user to prefer using any group of devices together. Certainly, the higher the probability (i.e. the joint probability), the more preferable the corresponding group of devices would be to support any specific activities.

Bayesian networks can be constructed based on historical data regarding devices used in various workflows functions. In fact, previously developed learning algorithms can learn the network structure (i.e. graph representation DAG) as well as the parameters (i.e. probability distribution P) from complete or incomplete data [153]. One approach is to find the network that maximises the likelihood of the data using searching algorithms [71]. In [191], Wu et al. successfully construct Bayesian Network using activity dataset [129] to accurately predict user activity. Similarly, Bayesian networks can be constructed from a dataset of historical workflows. Moreover, user historical workflows include the dependencies between the functions, which can be used to guide the structure learning of

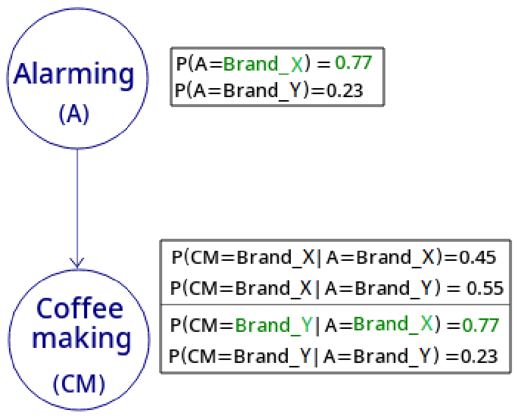

Alarming (A)
P(A=Brand_X) = 0.77
P(A=Brand_Y)=0.23
Coffee making (CM)
P(CM=Brand_X|A=Brand_X)=0.45
P(CM=Brand_X|A=Brand_Y) = 0.55
P(CM=Brand_Y|A=Brand_X) =0.77
P(CM=Brand_Y|A=Brand_Y) =0.23

Figure 4.2: User preference representation using Bayesian networks for the example in Figure 4.1. Based on the conditional probabilities, in green, the Bayesian network can infer that a user prefers to use $Brand\_X$ alarm device and $Brand\_Y$

the Bayesian network. User preference modelling is not the central focus of this paper. We rely on existing machine learning techniques and will not investigate this issue further.

4.2.3 Problem Formulation

Given a network model N, user preference model $M(F, D)$, and a workflow $W =< F, E >$, how to determining the set of devices $D_s \subseteq N$. Such that D_s fulfils all functions F in the workflow W and maximises user preferences.

We formulate this problem as following:

INPUT: $(N, W(F, E), M(F, D))$.

OUTPUT: $d \in D_s \subseteq N$

OBJECTIVE:

$$\begin{gathered}\text{argmax}_{D_i \subseteq N} M(F, D_i) \\ \text{subject to:} \\ \forall f \in F \, \exists d \in D_i, f \in Cap(d)\end{gathered} \tag{4.4}$$

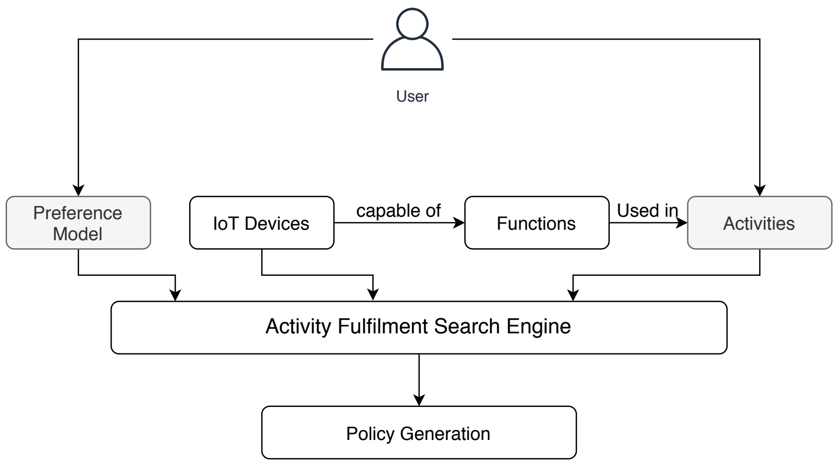


Figure 4.3: Automatic activity fulfilment using user preference.

The input is the network model N that represents all IoT devices in the smart space, workflow $W(F, E)$ that represents a given user activity, and user preference model $M(F, D_i)$ that returns a utility value to indicate how user prefer to use devices D_i for functions F. The goal is to find the set of devices D_i that satisfies all activity functions F and have the maximum preference value returned by $M(F, D_i)$

This optimisation problem will be solved using searching algorithms. Searching algorithms will be used to search the set of devices that maximise user preferences the subset of devices in the network N that satisfy the flow requirements F. Such that for any set of devices D_i a query will be sent to the preference model $M(D_i, F)$ to get the associated preference probability score. Then the devices with the highest score will be selected to implement the workflow functions.

4.2.4 Activity Fulfilment Searching Methods

Three common optimisation algorithms have been used to tackle the search problem in Equation (4.4). Hill Climbing (HC) and Simulated Annealing (SA) [107] are selected to represent local optimisation algorithms, and

Genetic Algorithm (GA) is selected to represent global optimisation algorithms [14]. The Brut-Force search also used as a baseline and also to investigate what extend it can scale. The aim is to test which algorithm is more suitable to solve this problem in terms of scalability, efficiency and solution quality.

As a simple search technique, hill climbing starts with an arbitrary selection of devices and gradually improve user preferences by changing some of the devices with alternative ones. Although hill climbing is highly efficient, it can be easily trapped in local optima and may not scale well to large problems. In case this problem includes local optima, simulated annealing is also used, which can escape the local optima by allowing less optimal movement based on a probability function. On the other hand, the genetic algorithm is a well-known evolutionary computing algorithm and has been widely demonstrated to perform well on many difficult combinatorial optimisation problems. Finally, the brute-force algorithm is systematically enumerating all possible candidates for the solution and selecting the best candidate. It is guaranteed to find the optimal solution. However, it cannot scale to a large problem.

The device selection Algorithm 1 takes a network N, a user preference model M, and a workflow W as inputs and outputs a set of devices D_s and their user preference score $Score_s$. The output devices D_s must satisfy the workflow W functional requirements, such that the higher user preference score $score_s$ they have the better (this depends on the searching algorithm, e.g. GA, HC, or SA).

The selection Algorithm 1 works as follows:

1. Selects all network devices D_{cands} that satisfy workflow requirements (i.e. F) as shown in Equation (4.5). This eliminates devices that are not required to satisfy workflow functions F.

2. The selected devices D_s is initialised with random device candidates and its preference score is calculated.

3. Each of the searching algorithms then performs the following three steps until its termination condition is met. The termination condition of each searching algorithm will be discussed later.

4. Selects a new candidate solution $D_i = alg(D_{cands}, F)$

5. Calculates the new candidate solution D_i score using the preference model (i.e. $M(D_i, F)$)

6. Based on the preference score, the existing candidate D_s is either replaced by the new one D_i or not.

$$\forall d \in D_{cands} \subseteq N, f \in F : \; f \in Cap(d) \tag{4.5}$$

Algorithm 1: Device selection searching algorithm

```
  Input: $N, W, M$
  Output: $score_s, D_s \subseteq N$
1 $D_{cands} = candidate\_selection(N, F)$ ;
2 $D_s = getRandomCand(D_{cands}, F)$ ;
3 $score_s = M(D_s, F)$ ;
4 while not termination condition do
5     $D_i = search_{alg}(D_{cands}, F)$ ;
6     $score_i = M(D_i, F)$ ;
7     $score_s, D_s = search_{alg}(D_i, D_s)$ ;
8 end
```

4.2.5 Evaluation

Empirical evaluation has been carried out to determine which of the four searching algorithms best suits our device selection problem, presented in Equation (4.4). The solution (i.e. the best devices to fulfil the activity)

Table 4.1: Genetic algorithm hyper-parameters

Parameter	**Value**
Generations	1000
Population size	$\|F\|$ * 200
Crossover rate	0.7
Mutation rate	0.2
Elitism rate	0.1
Selection Method	Tournament
Tournament size	3

Table 4.2: Simulated annealing hyper-parameters

Parameter	**Value**
Steps	200000
Max temperature	0.2
Min temperature	0.0001

quality is determined using two evaluation metrics; the user preference model utility value for the selected devices.

Experimental setup

The hyper-parameters for genetic algorithm and simulated annealing have been empirically fine-tuned. It has been found that the parameter settings summarised in Table 4.1 and 4.2 enable the two algorithms to perform reasonably well.

The searching algorithms are compared in terms of scalability and time efficiency by using multiple workflows composed of four to seven functions (i.e. $|F| = 4, 5, 6, 7$), where each function has seven alternative devices in the network N, see Table 4.3. Note the search space is $|F|^7$.

A user preference model is constructed as Bayesian networks where each node represents a function and its values are the alternative devices.

Table 4.3: Experimental Settings

No Workflow functions $\lvert F\rvert$	4, 5, 6, 7
Optimal user preference $P(D_s)$	0.34, 0.42, 0.3, 0.24
No Network Devices $\lvert N\rvert$	$rand(\lvert F\rvert, 4 * \lvert F\rvert)$
No Device capabilities	$rand(2, 7)$
No alternative devices	7
Experiment runs	30

To verify the effectiveness of the searching algorithms, we mimic user preferences by tuning the Bayesian networks probability distribution. We do that by randomly selecting a set of devices to be preferred by the user and assign high joint probability value p for them. This is achieved simply by setting all conditional probabilities related to the desired collection of devices to be $p^{1/|F|}$ so that the joint probability will be p. For example, in Figure 4.2, to mimic that a user prefers to use Brand_X alarm device and make coffee using Brand_Y, the joint probability should be greater than any other alternatives (e.g. 0.6). For that to be the result, alarm Brand_X and coffee maker Brand_Y conditional probabilities have to be $0.6^{1/2}$; hence, their joint probability $P(CM = Brand\_Y, A = Brand\_X) = 0.6$. Table 4.3 shows the generated optimal user preference probability $P(D_s)$ to $(0.34, 0.42, 0.3, 0.24)$ for each composed workflow of $4, 5, 6 and 7$ functions respectively. These values represent the optimal solution and used to evaluate the searching algorithms.

Results and Discussion

The results show that the hill-climbing algorithm, as expected, is often trapped in local optima solutions. It does not select devices for the workflow function that are most preferred according to the preference model. The effectiveness of the hill-climbing algorithm deteriorates sharply with the increasing number of workflow functions, as shown in Figure 4.5. Sim-

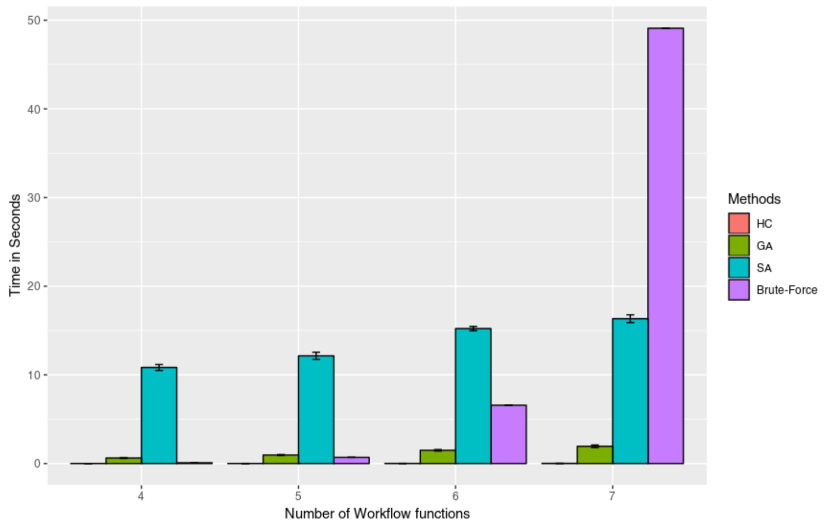


Figure 4.4: Time efficiency comparison

ulated annealing and genetic algorithm both can always find the optimal solution, see Figure 4.5. We verify the optimality of the solution using the known user optimal preference probability shown in Table 4.3.

The searching algorithms vary in terms of the time they take to reach the solution, as shown in Figure 4.4. Hill climbing algorithm is the most efficient algorithm in terms of elapsed time. However, its effectiveness is not satisfactory as it is not finding the optimal set of devices that maximise user preference probability, see Figure 4.5. Simulated annealing outperforms Hill climbing in terms of effectiveness, but at a high cost of computation efficiency, since it clearly requires significantly longer computation time. However, simulated annealing still better than Brute-Force search, which shows an exponential increase in the time it takes with the increase of a number of functions in a workflow.

The best balance between effectiveness and efficiency can be achieved in our experiments by using a genetic algorithm. The genetic algorithm produces an optimal solution like simulated annealing and Brute-Force, and it scales better, as shown in Figure 4.4. Although Brute-Force is ef-

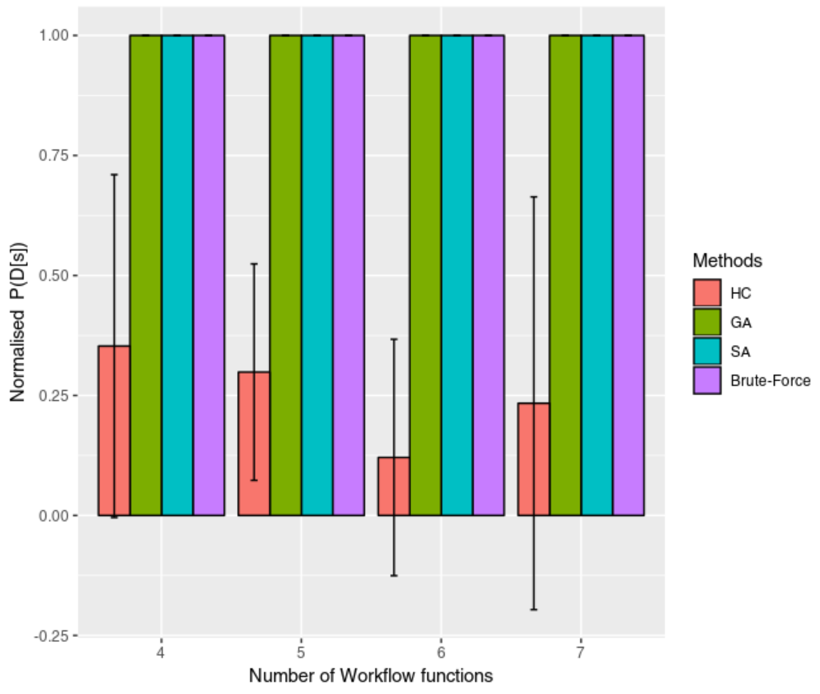


Figure 4.5: User preference optimisation comparison

ficient in small problems, it fails to scale as the genetic algorithm. For example, in Figure 4.5 Brute-Force is more efficient than the genetic algorithm for problems with a workflow of 4 or fewer functions, however, for a workflow of five or more functions the time it takes increases exponentially.

4.3 Automatic Policy Generation

In the previous section, we discussed the algorithms that can automatically select devices to fulfil user activities while considering their preference. In association with the automatic activity fulfilment, network access policy need to be generated to support the activity devices and enforce the least privilege principle. We focus on network access control and assume authentication, such as key-based authentication [49], is supported in the network. Automatically generated policies can be subsequently enforced by any existing enforcing mechanisms developed for IoT such as Software Defined Networking [15].

The idea is to derive network ACLs policy for the workflow selected devices D_s to fulfil the network requirements R, in Equation (4.2), for each device to execute the assigned function. Note that the generated policy is correct by design as it is generated to satisfy the activity workflow.

Network requirements can include destination IP addresses, transport protocol, and port numbers, bandwidth, duration. We consider the source and destination IP addresses, destination port, and transport protocols. These requirements can be easily mapped to ACL policy P, as shown in Algorithm 2. The generated policy guarantees that the selected devices D_s can perform the activity and obey the least privilege principle. The ACL policy can be easily enforced within and on the border of the network using the framework, we proposed in Chapter 3.

The policy generation in Algorithm 2 tasks a workflow W, the selected devices D_s, and the network requirements Net_{req} as an input and returns a policy P as a set of rules. The algorithm creates a rule for each edge in the workflow DAG graph. For example, for a directed edge from f_1 to f_2 the algorithm will create a rule that allows the device that executes f_2 to connect to the device that executes f_1. It gets the pair of devices that fulfil every function f_i in the workflow and the functions f_j that has an edge to it, parents, (lines 2-5). Then it retrieves their IP addresses (line 6, 7) and the network requirements (the ports and transport protocol) for the child function (line 8). Finally, it creates a rule to allow the devices that fulfil the child and parent to communicate using the required port and transport protocol (lines 9-11).

Algorithm 2: Policy Generation Algorithm

```
   Input  : $W < F, E >$, $D_s$, $Net_{req}$
   Output: Policy $P$
1  Set $policy \leftarrow [\,]$ ;
2  foreach $f \in F$ do
3      foreach $f_j \in Parents(f_i)$ do
4          $d_j \leftarrow get\_device(D_s, f_j)$ ;
5          $d_i \leftarrow get\_device(D_s, f_i)$ ;
6          $dstIP \leftarrow get\_IP(d_j)$ ;
7          $srcIP \leftarrow get\_IP(d_i)$ ;
8          $dstPorts_protos \leftarrow get\_net\_req(Net_{req}, f_i, d_i)$ ;
9          foreach $(dstPort, proto) \in dstPorts_protos$ do
10             $rule \leftarrow \{src\_ip = srcIP, dst\_ip = dstIP, dst\_port =$
                $dstPort, tp\_proto = proto\}$ ;
11             $policy+ = rule$ ;
12         end
13     end
14 end
```

4.4 Conclusion

Activity workflows are a common technique for specifying automation activities in smart environments. In existing systems, users have to specify the underlying devices to execute activities. We extended this by allowing users to specify abstract workflows that are instantiated for the particular environment. Previous research has not explored the automatic generation of concrete workflows and enforcement of least privilege based on user requirements and preferences. We present an approach that decouples workflow requirements from the specific devices, enabling devices to be selected on deployment to meet user preferences. We also show how an automatic policy can be generated given the network requirements for executing a particular workflow function.

We formulated user activity automation as a constraint optimisation task and solved it using heuristic searching algorithms. Our experiments indicate that for a small IoT device network, brute-force search is a reasonably good choice for optimising device selection. However, for larger networks, the genetic algorithm provides the best results among the algorithms tested, as it yields a good balance between efficiency and effectiveness.

Chapter 5

Multi-User IoT Sharing Policy in Smart Spaces

5.1 Introduction

The previous chapters (3 and 4) presented approaches to use user input to derive access control for a single user system. However, it does not study the multi-user scenario, where multiple IoT devices owned and shared by different users co-exist in the same smart space. The current single-user system requires users to either share their devices with full access or none [29, 200]. Sharing devices with full access violates the principle of least privilege and is a high security risk [26]. Instead, it is more sensible to share under certain restrictions [198], depending on the level of trust among the users involved; hence, limiting the risk of abuse.

This chapter focuses on the problem where multiple devices are owned by different users who want to share their devices under certain conditions. For secure sharing, a policy language is required to enable users to express their sharing policies. Existing policy languages mainly take into account environment context (e.g. location) for access control [160, 198]. A more relevant policy EXPAT [193] focuses on capturing users expectation on how their devices should work, for example, if a user is not at home, the front door should be locked. These policy languages don't support controlled device sharing at the level of individual device functionality. Therefore, we develop a new Policy Language, called Sharing Policy Language (SPL) described in Section 5.6, to allow devices owners to specify precisely the conditions in which they are willing to sharing their device functionality. For example, users can specify who can use their devices, what functions their devices can be used for, and what devices can interact with their devices.

This chapter address the IoT devices sharing from two perspectives the device owner perspective and another from the device user prospective. The first aspect is how to enable fine-grain IoT sharing such that users can share a device for a specific function with a specific user. The second aspect is how to automatically fulfil users activities using the most secure set of devices from the available shared ones. This chapter addresses this problem by designing a secure sharing system, Section 5.5, supported by a sharing policy language, Section 4, and formulating the sharing problem as an optimisation problem, Section 5.4.

5.1.1 Chapter Goals

Motivated by the absence of a multi-user sharing system for smart space, a novel IoT architecture and a newly designed sharing policy language are developed. Moreover, motivated by the success of optimisation methods on finding the optimal solution subject to a set of constraints, the IoT sharing problem is formulated as Integer Linear Programming (ILP) problem. The aim is to optimise the set of devices that fulfil users activities to the most secure available ones subjected to a set of constraints that represents users' sharing policies. In particular, this chapter aims at addressing the following objectives:

- Designing a novel multi-user IoT Secure Sharing (IoTSS) system that enables smart space users to share and use IoT devices in a secure manner.

- Designing a new IoT sharing language that allows users to specify fine-grained sharing conditions and demonstrate that they are always be obeyed while sharing devices.
- Mathematically formulate IoT sharing to enable translating it from network domain to optimisation domain.
- Develop a translating method to translate IoT sharing from network domain into an ILP problem.
- Investigating the scalability and performance of ILP solvers on various IoT sharing problem instances.

5.1.2 Chapter Organisation

The remainder of this chapter is organised as follows: The next discusses the threat model in IoT sharing environment. Section 5.3 illustrate the IoT sharing problem using a smart home scenario. Section 5.5 presents an overview of a novel IoT secure sharing system. Section 5.4 describes the multi-user IoT sharing as optimisation problem. Section 5.6 presents the sharing policy language. Section 5.7 introduces IoTSS engine and how it formulates the IoT sharing as an ILP problem. Section 5.8.1 illustrate the the experiment setup, and Section 5.8.2 discuss the results. Finally, Section 5.9 concludes this chapter and its findings.

5.2 Threat Model

The smart space threat model is illustrated in Figure 5.1. The trust worthy components are the owner of the IoT devices, the IoT hub and the switch. The rest of smart space components (i.e. other users and other IoT devices) are considered to be untrustworthy as per previous research [73, 17, 172, 101] and [198, 58, 62] as explained in Section 2.2.1 and 2.2.3.

This work focuses on mitigating two threats when IoT devices are shared in smart spaces: (a) unauthorised access to an IoT device without permission of the device owner; and (b) the risk of using devices with known vulnerabilities. In (a) IoT devices' owners share their devices with other users based under certain conditions, but without any access control in place, usually, users share IoT devices based on trust between them[58]. For example, a user may share his/her *Speaker* to be used for any function except for the purchasing function. However, without an enforcement mechanism to control how shared devices should be used, users can abuse the shared devices, resulting in a loss of security such as confidentiality or availability. To mitigate this, we develop a sharing policy language to enforce owners sharing policies. In (b), our aim is to reduce the use of vulnerable shared devices where a more secure alternative exists. We tackle (b) by formulating device sharing in the form of an optimisation problem to use the most secure devices to automate users' activities without violating any security policies.

5.3 IoT Sharing Scenario

This section illustrates the IoT sharing problem using a smart home as a typical smart space and motivates the need for developing a new sharing policy language.

Figure 5.2 shows smart home devices owners *Jack* and *Alex*, and the user *Others* that refers to their friends/guests. Table 5.1 shows the available devices with the corresponding functions each device supports, device security score, and the owner. Additionally, each device's function limit identifies the number of activities that can use a device function concurrently. For example, Jack's TV security score is 0.7 and its *display* function can only be used by a single activity at a time, whereas its *find* function

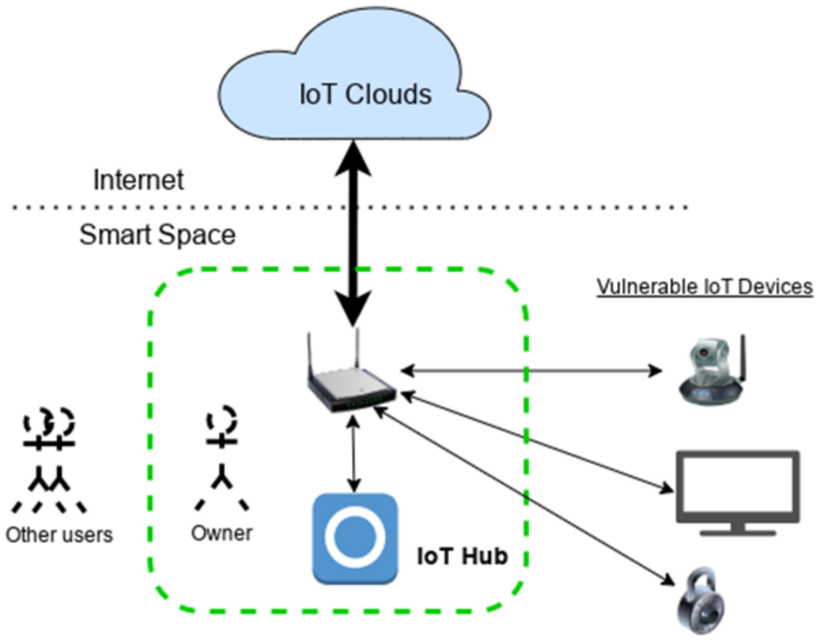


Figure 5.1: Mutli-user IoT sharing threat model. The green box contains the trusted components of the shared smart space.

can be used by multiple activities at the same time. For the sake of illustration, we assume that the number of maximum activities that a device function can support concurrently is either 1 or 10. Devices belong to *Jack* or *Alex* or both of them, as shown in Table 5.1. Therefore, *Alex* and *Jack* define device their sharing policies, as shown in Table 5.3, for sharing their devices with users *Others*.

Given the scenario above, *Alex* and *Jack* want to share their devices with each others and with other users (i.e. guest and friends). They want to define a) with whom they are sharing, b) what device, and c) what function the device should be used for. Moreover, they may also want to not allow their shared devices to be used with other devices in the same activity. Therefor, they defined their sharing policies as shown in Table 5.3 as follows:*Jack* would like to share his devices under the following policy *"do not share my speaker with coffee machine, do not share my TV with anyone*

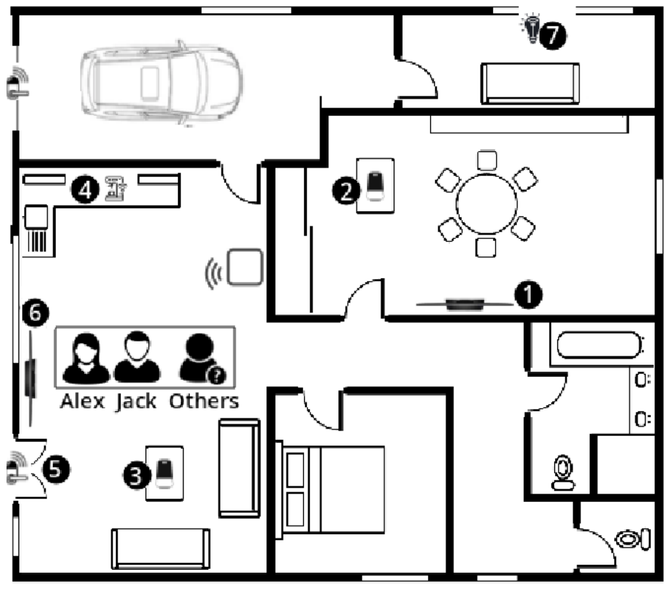


Figure 5.2: IoT sharing in a smart home scenario. Jack owns devices 1 and 2, Alex owns devices 3 and 4, and both as family members own the rest of the devices. Devices' functions and security score are presented in Table 5.1

except Alex, and don't share my speaker if the fridge is used.". Alex, on the other hand, would like to share her devices with the following policy *"do not share my speaker find function with anyone except Jack, do not share door or garage lock with anyone except Jack"*. Table 5.3 shows these policies with default to share policy and also with default not to share.

Given the smart space scenario in Figure 5.2 and users' sharing policies in Table 5.3, users (*Alex, Jack* and a guest, *Other*) define the following activities: *Jack* would like to automate a movie time activity as follows: *"find for a movie, display it and turn off Light"*. Likewise, *Alex* would like to automate her back home activity which opens the door, prepare a coffee, find selected music and play it, as follows: *"unlock the main door, prepare a coffee and find and play music"*. Lastly, a guest wants to define activity to run music as follows: *"find and play music"*. The required functions for

Table 5.1: Smart home devices functions, security score and owners

Device	Functions	Limit	Sec. Score	Owner
d_1- TV	f_1-Find	10	0.7	Jack
	f_2-Display	1		
	f_3-Play	1		
d_2- Speaker	f_1-Find	10	0.75	Jack
	f_7-Buy	1		
	f_3-Play	1		
d_3- Speaker	f_1-Find	10	0.8	Alex
d_4- Coffee Machine	f_6-Coffee	1	0.5	Alex
	f_3-Play	1		
d_5- Door Lock	f_5-lock/unlock	1	0.7	Both
d_6- TV	f_1-Find	10	0.6	Both
	f_2-Display	1		
	f_3-Play	1		
d_7- Light	f_4-Light On/Off	1	0.65	Both

each activity is summarised in Table 5.2. The question now is which of the available shared devices should perform these functions to maximise the total security score and without violating any user sharing policies. Section 5.4 formally defines this problem as a sharing problem.

Table 5.2: Users activity automation as set of functions

ID	Activity functions	User
A_1	(Find, Display, Light)	Jack
A_2	(D-lock, Coffee, Find, Play)	Alex
A_3	(Find, Play)	Guest

Table 5.3: User policies when the default policy is *share* and *do not share*.

Default	Share
Jack	- Don't share Speaker Buy function with anyone. - Don't share TV except with Alex.
Alex	- Don't share Speaker with Coffee machine. - Don't share Doorlock except with Jack.
Default	**Do not share**
Jack	- Share Speaker's find and play functions with anyone. - Share TV with Alex.
Alex	- Share Speaker except with Coffee machine. - Share Doorlock with Jack. - Share Light, Coffee Machine and TV with everyone.

5.4 IoT Sharing Problem

In a smart space, multiple users intend to share their IoT devices, such that they can use these shared devices to fulfil their activities without violating any device sharing policies. We call this IoT sharing problem. We assume users are willing to share their devices under some predefined sharing policies [45].

In this problem, we consider a set of users $\mathcal{U}$ who share a set of IoT devices $\mathcal{D}$ under a predefined set of sharing policies $\mathcal{P}$. These devices are capable of supporting a set of functions $\mathcal{F}$. Users define their activities $\mathcal{A}$ abstractly as a set of required functions. These activities have to be fulfilled by a set of IoT devices without violating user sharing policies. We define key terms related to this problem in more details as follows.

Users $\mathcal{U}$: We classify users into devices' owners and ordinary users. The devices' *owners* can define *sharing policies* on the devices they want to share with other users. Ordinary users compose their *activities*, abstractly, as a set of functions to be fulfilled by available shared devices.

IoT devices $\mathcal{D}$: We consider the smart space as a collection of *IoT devices*

$\mathcal{D}$ where each device $d \in \mathcal{D}$ has the following attributes: (a) it supports a set of functions given by $Cap(d)$ where each function $f \in Cap(d)$ can be used by a limited number of concurrent activities given by $Limit(f, d)$; (b) each device has a security score $Sec(d)$ which is a numerical value indicating the security score of device d (the larger, the better); (c) it resides in a location in the smart space which can be retrieved by $Loc(d)$ function.

Available IoT functions $\mathcal{F}$: All available functions in the network are jointly determined by all functions supported by all available IoT devices as $\mathcal{F} = \cup_{d \in \mathcal{D}} Cap(d)$.

User activities $\mathcal{A}$: A user automation activity A represents a routine or daily activity that the user want to automate using IoT devices. Each activity $A \in \mathcal{A}$ is defined as a pair $A = (F_A, u_A)$ of the set of required functions F_A and u_A represents the user this activity is running on behalf of him/her.

Sharing Policies $\mathcal{P}$: A sharing policy $P \in \mathcal{P}$ is a set of rules that defines the situations in which the owner's of the devices want to share or not to share their devices. Each owner user can define a sharing policy $P \in \mathcal{P}$ on his/her devices.

To help formulate our IoT sharing problem mathematically, we introduce the following definitions.

Definition 1. *An activity fulfilment I_A of an activity $A = (F_A, u_A)$ is defined as a set of function and device pairs (f, d) where $f \in F_A$, and $d \in \mathcal{D}$, subject to:*

$$\forall f \in F_A \exists f \in Cap(d), (f, d) \in I_A$$
$$\forall (f, d), (f', d') \in I_A, \text{ if } (f = f') \rightarrow (d = d')$$

Which means that each function $f \in F_A$ is fulfilled by exactly a single device $(f, d) \in I_A$.

It is common in a smart space to have more than one possible activity fulfilment for a given activity as multiple devices can support a given function. We denote all possible fulfilment of activity A as $\mathcal{I}_A$. Hence, the overall fulfilment for all activities $\mathcal{A}$ is given by $\mathcal{I}_{\mathcal{A}} = \{\mathcal{I}_A | A \in \mathcal{A}\}$. Note that $\mathcal{I}_{\mathcal{A}}$ represents the search *space of all possible fulfilment of activities $\mathcal{A}$ including those that violate sharing policies or using less secure devices.*

Definition 2. *A single activity fulfilment of a user activity $A \in \mathcal{A}$ is I_A, which belongs to a set of available candidates $\mathcal{I}_A$. Hence, a solution candidate for our IoT sharing problem is a set of all activities $\mathcal{A}$ fulfilment $\mathbb{I}_{\mathcal{A}} = \{I_A : A \in \mathcal{A}\}$ where there is an activity fulfilment I_A for each activity $A \in \mathcal{A}$.*

Definition 3. *The security score of a solution candidate $\mathbb{I}_{\mathcal{A}}$ is the sum of the security scores of all devices which have been selected, which is calculated as below.*

$$S(\mathbb{I}_{\mathcal{A}}) = \sum_{(f,d) \in I_A, I_A \in \mathbb{I}_{\mathcal{A}}} Sec(d)$$

Definition 4. *The function $C(\mathbb{I}_{\mathcal{A}}, d, f)$ counts the number of times function f of device d has been used in $\mathbb{I}_{\mathcal{A}}$.*

Definition 5. *A sharing policy P is a pair (u_p, R_p) where u_p refers to the policy owner and R_p stands for the policy rules. Each rule $r \in R_p$ is a function of a potential solution $\mathbb{I}_{\mathcal{A}}$ such that $r(\mathbb{I}_{\mathcal{A}})$ verifies that $\mathbb{I}_{\mathcal{A}}$ does not violate the rule r.*

A sharing problem that consists of a set of user activities $\mathcal{A}$ that need to be fulfilled by a subset of a set of devices $\mathcal{D}$, where $\mathcal{D}$ is shared under a set of policies $\mathcal{P}$, this problem can be formulated as following.

$$\begin{aligned}
&\textbf{INPUT} : (\mathcal{I}_{\mathcal{A}}, \mathcal{P}, \mathcal{D}) \\
&\textbf{OBJECTIVE} : \mathbb{I}^*_{\mathcal{A}} = \operatorname{argmax}_{\mathbb{I}_{\mathcal{A}}} S(\mathbb{I}_{\mathcal{A}}) \\
&\text{s.t.} \\
&\forall P \in \mathcal{P}, \forall r \in P, r(\mathbb{I}_{\mathcal{A}}, P) = True \\
&\forall d \in \mathcal{D}, f \in Cap(d), C(\mathbb{I}_{\mathcal{A}}, d, f) \leq Limit(d, f)
\end{aligned} \tag{5.1}$$

The goal of this formulation is to maximise the overall security score of the devices (i.e. the objective function $S(\mathbb{I}_{\mathcal{A}})$) that fulfil the activities $\mathcal{A}$ subject to two constraints. The first constraint is that no activity fulfilment in a potential solution $I_A \in \mathbb{I}_{\mathcal{A}}$ violates any of the sharing policies $\mathcal{P}$. The second constraint ensures that no device's function is used in activities fulfilment more than its concurrent support limit.

5.5 IoT Secure Sharing Architecture

In this section, we discuss the high-level of the IoT Secure Sharing (IoTSS) architecture and how we model smart space components to enable IoT secure sharing. Figure 5.3 depicts the main components of IoTSS architecture.

Smart space is equipped with IoT devices that support different functions that are utilised by users. Users express their activities as a set of connected functions that fulfilled the desired task. These functions will be fulfilled by a set of shared IoT devices. The use of smart space devices is governed by sharing policies specified by devices' owners. At any time, device usage in any user activity must not violate any sharing policy. Moreover, The IoT devices varied in their security. For example, the less number of vulnerabilities a device has, the more secure it is. Besides enforcing IoT sharing policies, it is required that user activities are fulfilled by the most secure set of devices to increase the overall smart space security.

IoTSS works as follows: *users* build their activities as a set of *functions* that eventually fulfilled by available shared *devices*. These devices are owned by a subset of users (i.e. owners) who share them under sharing policies. The owners specify their sharing policies based on: (a) who are the authorised *users* they want to share their devices with; (b) which device *functions* they want to share; (c) which *devices* are allowed to interact with their devices (i.e. be used in the same activity with their devices). The goal is to fulfil activity functions requirements using the most secure set of devices without violating sharing policy. This problem is formulated as an ILP problem to be solved using ILP solver. Once the ILP solver select which device should execute which activity function, it passes this concrete activity to the hub and the PEP. The hub will operate the activity while the PEP will enforce the sharing policies to ensure that what users see in the application layer is actually enforced by access control.

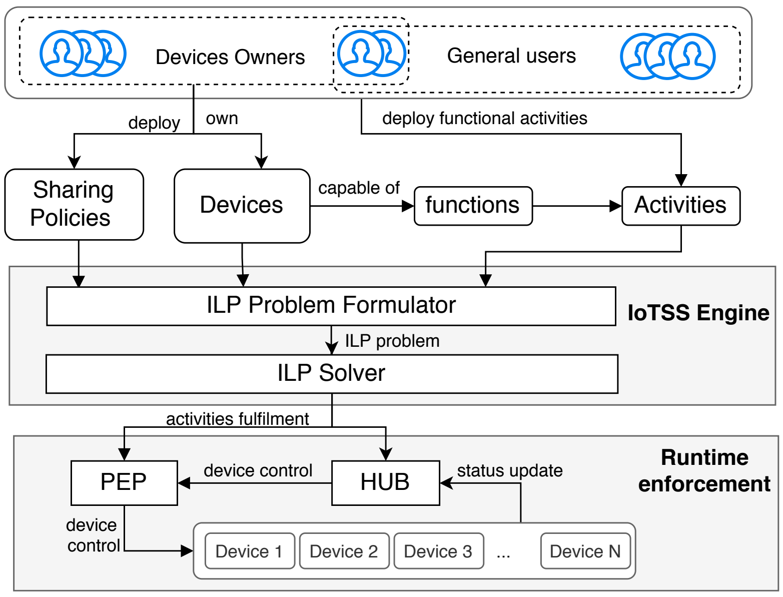


Figure 5.3: IoTSS System Architecture

IoTSS Engine. It converts IoT sharing from network domain into ILP domain and use ILP solver to solve it. IoTSS engine consists of (1) *ILP Problem Formulator* which translates available devices, user activities and sharing policies into decision variables, and then it uses these decision variables to generate ILP constraints and objective function for the corresponding ILP problem; (2) *ILP Solver.* The ILP solver uses off-the-shelf ILP solvers to solve the ILP problem generated by the ILP problem formulator. Particularly, it finds the most secure set of devices to fulfil user activities while satisfying users' sharing policies. The output specifies for each activity which device should be used to fulfil each of its functions. Then activities are deployed using the selected devices and installed in the Hub, see Figure 5.3.

Policy Enforcement Point (PEP). The PEP [174] is the system component that guards access to the resources (IoT devices) based on access policies. The PEP can enforce the users' sharing policies in two levels: (a) in the network level, by allowing or denying network access between IoT devices and the hub or between the devices themselves; (b) in the application layer, by allowing or denying access to certain function (through, e.g. API calls). The network-level access control enforcement can take place in a switch (e.g. OpenFlow-enabled switch). For example, in a hub-based smart space, the hub orchestrating inter-device interactions can be supported by only allow devices to talk to the hub and block all device-to-device communications. (Note that enforcing devices communications through the hub will not affect devices functionalities as it has been found in [70]). The application-level access control can be enforced by implementing the PEP as a proxy that set between the hub and the rest of the devices such that it intercepts hub to IoT control messages and only allows the hub to control devices that are selected by the ILP solver.

Note that although the Hub and PEP receive activities fulfilment devices set, they do different roles. The Hub is responsible for the operation of the activities, while PEP is enforcing users' sharing policy at runtime. This design keeps operation and policy enforcement independent and can be updated or changed without affecting each other.

5.5.1 Practicality

The proposed IoTSS system can be integrated into any smart space controller (e.g. Mozilla WebThings Gateway[121], Google home [66], Samsung SmartThings [84]). The devices, as well as their capabilities, can be retrieved from the hub. For example, Mozilla Gateway implements Web of Things that uses a semantic language to describe the things and their capabilities. Hence, it can be retrieved either directly from the devices in the network or from the Mozilla Gateway. Google home [66] defines the devices' capabilities using what is called traits. The IoT devices security score can be calculated using the device vulnerabilities' score, which can be retrieved from a vulnerability dataset like Common Vulnerability Scoring System (CVSS) [117]. For example, a speaker IoT device with larger total number of CVSS score is considered to be less secure than another speaker with less total number of CVSS score. Hence, the main inputs to the IoTSS can be retrieved from existing smart space technologies.

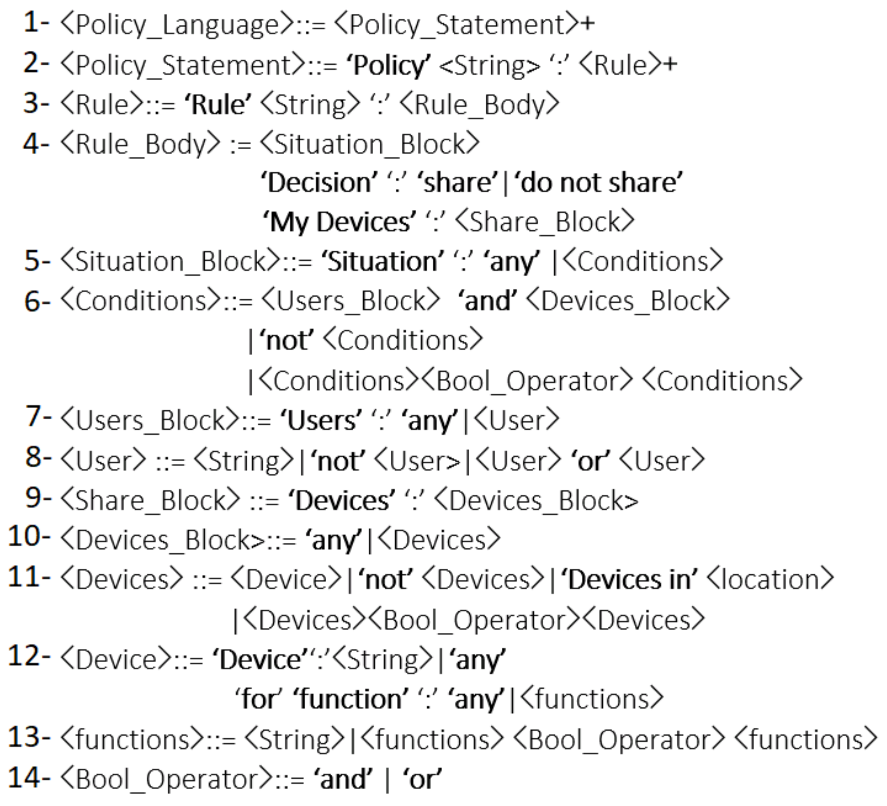
The PEP can be implemented in a switch to enforce network layer access control and as proxies for each IoT devices to enforce application/operation (function-based) access control. Policies can be enforced locally for local connection between IoT devices or at the edge of the network for external connections. The external connection includes services that the hub (where user activities are running) may need to connect as part of user activities. It also includes IoT devices connections to their manufacturer cloud.

5.6 Sharing Policy Language

Motivated by the scenario in Section 5.3, this section presents a novel sharing policy language to enable users to define their sharing policy using a high-level language. Figure 5.4 presents the concrete syntax of the Sharing Policy Language (SPL) as a Backus Normal Form (BNF) grammar [157]. SPL allows devices' owners to define what they want to share (i.e. what devices and functions to share) and in which context (i.e. with which users and peer devices or functions). This language inspired by user expectation language [193] and users sharing behaviour papers [58].

An SPL policy consists of one or more *policy statements* (line no. 1), each of which has a string label *Policy_Statement* as an identifier $P \in \mathcal{P}$. A policy statement (line no. 2) composes of a sequence of unordered rules that define the situations where a user wants to make a sharing decision (i.e. to share or not to share). Note that unlike firewalls rules where first match block wins, an activity fulfilment I_A must satisfy all sharing policy rules.

A *rule* (line no. 3 and 4), labelled by a string identifier, captures a specific *situation*, via *Situation_Block*, where a user wants to specify a shar-



```
1- <Policy_Language>::= <Policy_Statement>+
2- <Policy_Statement>::= 'Policy' <String> ':' <Rule>+
3- <Rule>::= 'Rule' <String> ':' <Rule_Body>
4- <Rule_Body> := <Situation_Block>
                  'Decision' ':' 'share'|'do not share'
                  'My Devices' ':' <Share_Block>
5- <Situation_Block>::= 'Situation' ':' 'any' |<Conditions>
6- <Conditions>::= <Users_Block>  'and' <Devices_Block>
                  |'not' <Conditions>
                  |<Conditions><Bool_Operator> <Conditions>
7- <Users_Block>::= 'Users' ':' 'any'|<User>
8- <User> ::= <String>|'not' <User>|<User> 'or' <User>
9- <Share_Block> ::= 'Devices' ':' <Devices_Block>
10- <Devices_Block>::= 'any'|<Devices>
11- <Devices> ::= <Device>|'not' <Devices>|'Devices in' <location>
                 |<Devices><Bool_Operator><Devices>
12- <Device>::= 'Device'':'<String>|'any'
                 'for' 'function' ':' 'any'|<functions>
13- <functions>::= <String>|<functions> <Bool_Operator> <functions>
14- <Bool_Operator>::= 'and' | 'or'
```

Figure 5.4: Sharing Policy Language syntax as a BNF

ing decision (*share, do not share*) on his/her devices that are defined in *Share_Block*. In other words, each rule can be read as following *"when situation holds, apply my sharing decision on my sharing block devices"*. For example, a user can define the following rule *"in a situation where the activity user is Jack do not share my device speaker for a function buy"*, as the speaker rule in Figure 5.5.

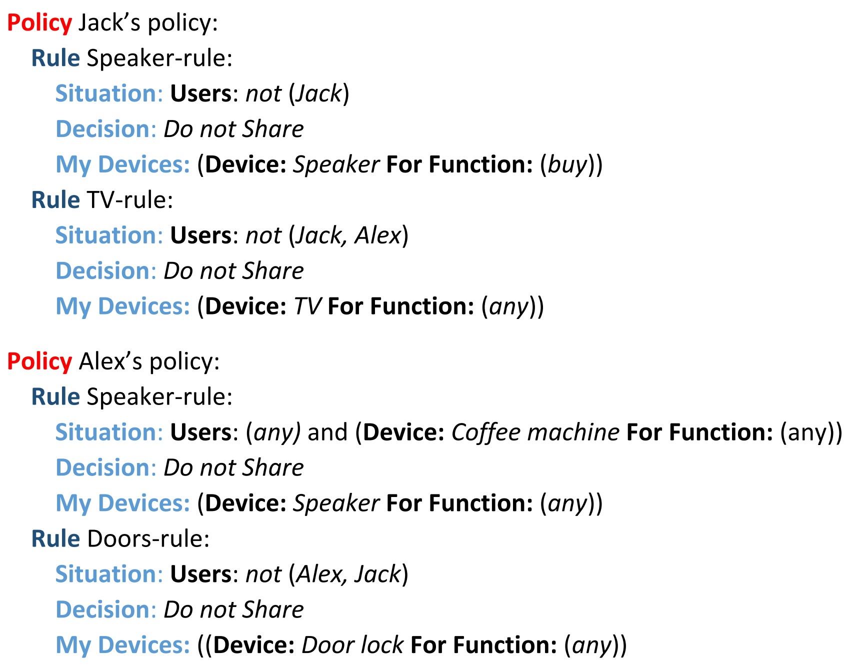
The *Situation_Block* (line no. 5) contains the conditions that describe the situation that will trigger the rule. If the situation conditions are met, then the sharing decision in *Share_Block* will be enforced. The situation condition can represented by the built-in **"any"** keyword which matches any situation. Also, satiation conditions can be represented by the *Conditions* expression.

The *Conditions* (line no. 6) are boolean expressions with the logical operators (i.e. and, or, not) to define a condition based on *Users_Block* and/or *Devices_Block*. *Users_Block* (line no. 7) starts with "Users:" label and defines a situation based on users. It can be either **"any"**, which match any user, or a specific set of users using *User* block (line no. 8) which contains one or more atomic predicates (i.e. **username**) with **"not"** or **"or"** boolean operation (line no. 14). The *Devices_Block* (line no. 10) defines situations based on peer devices or functions that will be used with the device. For instance, a situation can be defined as *"in a situation where an activity uses device Y, do not share Speaker"*. *Devices_Block* (line no. 10) uses *Devices* (line no. 11) expression that can define any combination of devices using *Device* (line no. 12) expression connected with three logical operators (i.e. not, and, or). It can also define devices based on their location. For example, *"in any situation do not share Devices in the bedroom"*. A *Device* expression consists of **Device_name** atomic predicate and *functions* (line no. 13)expression separated by a colon ':'. The later allows users to build a complex function constraint. For instance, a user can define a device expression " if device d is used for function a or function b".

Figure 5.5 shows how SPL is used to express *Alex* and *Jack* sharing policies in Table 5.3. Each of *Jack* and *Alex* has his/her own policy named after his/her name (i.e. *Jack_policy* and *Alex_policy*) which includes sharing rules for his/her own devices. For example, *Jack's* first rule defines that for any user, *Jack* doesn't want to share his Speaker to be used for buy function.

5.7 IoT Secure Sharing Engine

IoT Secure Sharing (IoTSS) engine converts IoT sharing task into an ILP problem then uses an ILP solver to optimise the security score of the selected devices that fulfil user activities such that the sharing policies are realised. We achieve this by representing the IoT sharing as an ILP problem. The binary integer decision variable $X_{A,d,f}$ takes a value of 0 or 1 to



Policy Jack's policy:
  Rule Speaker-rule:
    Situation: **Users**: *not* (*Jack*)
    Decision: *Do not Share*
    My Devices: (**Device:** *Speaker* **For Function:** (*buy*))
  Rule TV-rule:
    Situation: **Users**: *not* (*Jack, Alex*)
    Decision: *Do not Share*
    My Devices: (**Device:** *TV* **For Function:** (*any*))

Policy Alex's policy:
  Rule Speaker-rule:
    Situation: **Users**: (*any)* and (**Device:** *Coffee machine* **For Function:** (any))
    Decision: *Do not Share*
    My Devices: (**Device:** *Speaker* **For Function:** (*any*))
  Rule Doors-rule:
    Situation: **Users**: *not* (*Alex, Jack*)
    Decision: *Do not Share*
    My Devices: ((**Device:** *Door lock* **For Function:** (*any*))

Figure 5.5: *Jack* and *Alex* sharing policy example

indicate that function f of device d is going to be used in activity A. The ILP objective function, described in equation (5.2), maximises the summation of devices security score $Sec(d)$ overall devices used in $\mathcal{I}_\mathcal{A}$ according to the problem formulation in (5.1).

$$\begin{aligned} & \max \textstyle\sum_{\mathbb{I}_\mathcal{A} \in \mathcal{I}_\mathcal{A}} \sum_{(f,d) \in \mathbb{I}_\mathcal{A}} Sec(d) \;\; X_{A,d,f} \\ s.t. & \\ & \forall P \in \mathcal{P}, \forall r \in P, r(\mathbb{I}_\mathcal{A}, P) = True \\ & \forall d \in \mathcal{D}, f \in Cap(d), \textstyle\sum_{A \in \mathcal{A}} X_{i,d,f} \leq Limit(d,f) \end{aligned} \tag{5.2}$$

The mapping between the constraints in (5.1) and the ILP constraints are explained as follows:

- The first constraint enforces users' sharing policies $\mathcal{P}$. For every user policy $P \in \mathcal{P}$ it checks if a given activities fulfilment $\mathbb{I}_\mathcal{A}$ doesn't violate any rule $r \in P$. Each rule can be represented as a set of constraints that specify what set of decision variables cannot be used together. For example, $X_{(A_1,d_1,f_1)} + X_{(A_1,d_2,f_2)} \leq 1$ represents a rule that specifies that in an activity A_1 (which runs on behalf of a user u_1) cannot use a device d_1 for a function f_1 with a device d_2 for a function f_2.

- Each function f offered by device d must not be used by a number of activities higher than its limit $Limit(d, f)$. This is enforced by the first constraint that sums all decision variables using device d for function f in all activities fulfilment $I_A \in \mathcal{I}_{\mathcal{A}}$ and set this sum less than the device function limit $Limit(d, f)$.

Therefore, by translating all user activity and policies into ILP domain, we can maximise the use of secure devices without violating users policies.

5.7.1 ILP Problem Formulator

ILP Problem Formulator (ILPPF) converts our IoT sharing problem, modelled in Section 5.4 from network domain into the ILP domain.

Decision variables $\mathcal{X}$. The set of decision variables $\mathcal{X}$ includes all possible decision variables of alternative devices that can support activities function. A binary decision variable $X_{A,d,f} \in \mathcal{X}$ takes a value 1 to indicate that a device d will be used to fulfil function f of activity A or 0 if not (i.e. $X_{A,d,f} \in \{0, 1\}$). The set $\mathcal{X}$ is defined in (5.3).

$$\mathcal{X} = \{X_{A,d,f} : A \in \mathcal{A}, f \in F_A, d \in \mathcal{D}, f \in Cap(d)\} \tag{5.3}$$

Mapping all decision variables $\mathcal{X}$ to $\{0, 1\}$ defines a given activities fulfilment $I_{\mathcal{A}}$ which only consider as a valid solution if it doesn't violate any constraints.

Activity fulfilment constraints $\mathcal{C}_{\mathcal{A}}$. Each activity's function should be fulfilled exactly by one device from the set of available devices that support it. These constraints are enforced using the set defined in equation

(5.4), which ensures that the sum of all decision variables that include function f of activity A should have exactly one decision variable its value is 1 (i.e. one device is selected to fulfil this function among all available devices $\mathcal{D}$).

$$\mathcal{C}_{\mathcal{A}} = \{\sum_{\forall d \in \mathcal{D}} X_{A,f,d} = 1 : A \in \mathcal{A}, f \in F_A\} \tag{5.4}$$

Sharing policy constraints $\mathcal{C}_{\mathcal{S}}$. A rule $r \in P, P \in \mathcal{P}$ is defined over an activities fulfilment $I_{\mathcal{A}}$. A rule checks if $I_{\mathcal{A}}$ respects the rule constraints or not. Our ILP formulation supports several primitive rules such that each of which is converted into a set of ILP constraints as shown below.

Given $X_{A,d,f}, X_{\hat{A},\hat{d},\hat{f}} \in \mathcal{X}$, the set of primitive rules and the corresponding ILP constraints are presented below:

1. "Don't share my device $<d'>$with user $<u'>$". This rule is enforced by constraints that set all decision variables that use device $<d'>$in a user $<u'>$activities (i.e. $u_A = u'$) to zero (means should not be used).

   $$\{X_{A,d,f} = 0 : d = d', u_A = u'\}$$

2. Don't share my device $<d'>$for function $<f'>$with user $<u'>$. This rule is enforced by constraints that set all decision variables that use device $<d'>$for function $<f'>$in a user $<u'>$activities (i.e. $u_A = u'$) to zero.

   $$\{X_{A,d,f} = 0 : d = d', f = f', u_A = u'\}$$

3. Don't share my device $<d'>$for function $<f'>$with any user. This rule is enforced by constraints that set all decision variables that use device $<d'>$for function $<f'>$, regardless of the user, to zero.

   $$\{X_{A,d,f} = 0 : d = d', f = f'\}$$

4. Don't share my device $<d'>$with device $<d''>$. This rule is enforced by constraints that set the sum of all decision variables that use device $<d'>$and $<d''>$to be less than one. Which means only one of

these decision variables value can be set to one (i.e. used) but not both of them.

$$\{X_{A,d,f} + X_{\hat{A},\hat{d},\hat{f}} \leq 1 : d = d', \hat{d} = d''\}$$

5. Don't share my devices in location $<z>$with any user. This rule is enforced by constraints that set all decision variables that use device $<d>$where it is located in $<z>$to zero.

$$\{X_{A,d,f} = 0 : Loc(d) = z\}$$

Table 5.4 shows the SPL language description of all the primitive rules discussed above. Through combining these primitive rules, we can allow users to express different sharing policies they have for smart space. Overall all sharing policies $\mathcal{P}$ are enforced by the corresponding $\mathcal{C_S}$ set of constraints which is given by equation below. For example, SPL rule no. 1 in Table 5.4 states that in a situation where user u' do not share device d'. The corresponding ILP constraint for this rule is $X_{A,d,f} = 0$ where $d = d'$ and $u_A = u'$, meaning do not use device d' for any activity of user u'.

$$\mathcal{C_S} = \{\cup_{I_A \in \mathcal{I_A}} r(I_A) : r \in R_p, p \in \mathcal{P}\} \tag{5.5}$$

Device functions constraints $\mathcal{C_F}$. A device should not be used to fulfil any specific function in a number of activities that is more than what it can support.

$$\mathcal{C_F} = \{\sum_{\forall A \in \mathcal{A}} X_{A,f,d} \leq Limit(d,f), f \in Cap(d)\} \tag{5.6}$$

Objective function O. The objective function in (5.7) maximises the sum of the product of the binary decision variable $X_{A,d,f} \in \mathcal{X}$ and the associated device's security score $Sec(d)$, as a coefficient.

$$O = \text{argmax}_{d \in \mathcal{D}} \textstyle\sum_{X \in \mathcal{X}} Sec(d) \cdot X_{A,d,f} \tag{5.7}$$

Table 5.4: The primitive rules using SPL and the corresponding ILP constraints.

Rule	**Corresponding Constraints**
1. **Situation**: Users: <u′> **Decision**: Do not share **My Devices**: Device: <d′>	$\{X_{A,d,f} = 0 \| d = d', u_A = u'\}$
2. **Situation**: Users: <u′> **Decision**: Do not share **My Devices**: Device: d′ for function: <f′>	$\{X_{A,d,f} = 0 \| u_A = u', d = d', f = f'\}$
3. **Situation**: Users: any **Decision**: Do not share **My Devices**: Device: d′ for function: <f′>	$\{X_{A,d,f} = 0 \| d = d', f = f'\}$
4. **Situation**: Users: any and Device: <d″> **Decision**: Do not share **My Devices**: Device: <d′>	$\{X_{A,d,f} + X_{\hat{A},\hat{d},\hat{f}} \leq 1 \| d = d', \hat{d} = d''\}$
5. **Situation**: Users: any **Decision**: Do not share **My Devices**: Devices: in <z>	$\{X_{A,d,f} = 0 \| Loc(d) = z\}$

The IoT sharing corresponding ILP problem is presented in (5.8). It uses the objective function O, in (5.7), subject to activity fulfilment constraints $\mathcal{C}_{\mathcal{A}}$ in (5.4), activity fulfilment constraints $\mathcal{C}_{\mathcal{F}}$ in (5.6), and sharing policy constraints $\mathcal{C}_\mathcal{S}$ in (5.5).

$$\begin{aligned}&O = \mathrm{argmax}_{d \in \mathcal{D}} \textstyle\sum_{X \in \mathcal{X}} Sec(d) \cdot X_{A,d,f}\\ &\text{s.t.}\\ &\mathcal{C}_\mathcal{A} \cup \mathcal{C}_\mathcal{F} \cup \mathcal{C}_\mathcal{S}\end{aligned} \tag{5.8}$$

5.7.2 ILP Problem Formulator Algorithm

ILP Problem Formulator is depicted in Algorithm 3. The inputs to the Algorithm are the set of activities $\mathcal{A}$, the set of user policies $\mathcal{P}$, and the set of IoT devices $\mathcal{D}$. The output is an ILP problem (i.e. objective function O and a set of ILP constraints $\mathcal{C}_\mathcal{A} \cup \mathcal{C}_\mathcal{S} \cup \mathcal{C}_\mathcal{F}$).

The Algorithm starts by initialising the decision variable set as in Equation (5.3) and the three sets of constraints (lines 1,2). Each activity $A \in \mathcal{A}$. To generate activity fulfilment and sharing policy constraints, it goes through each activity A in the activity set $\mathcal{A}$ (line 3). For every function f in an activity function F_A all decision variables that includes the activity A it generates a constraints to ensure that at exactly one device is fulfilling the function f (lines 4-6). Sharing policy constraints are generated by passing every activity fulfilment to every rule in every policy (lines 8-15). The device functions constraints generated for all functions used by the given activities $\mathcal{A}$ (lines 16-18). The objective function generated (line 19) to maximise the sum of the security score of every device associated with a decision variable. Finally, the objective function and the set of constraints that forms the corresponding ILP problem are returned (line 20).

The Algorithm works under the assumption that the default policy is to share all devices unless specified otherwise in the sharing policies. However, if the default sharing policy is *not to share* and the sharing rules define what to share, then the solutions that do not match sharing rules situation and shared devices should be denied. For that, to work the rule function at line 11 need to be inverted to reject activity fulfilment that does not match it. The rest of the Algorithm should work as-is.

Algorithm 3: Formulate ILP problem from devices sharing settings.

```
Input: $\mathcal{A}$, $\mathcal{P}$, $\mathcal{D}$
Output: ILP optimisation problem
1  Initialise ILP constraints $\mathcal{C}_{\mathcal{A}} \leftarrow \phi, \mathcal{C}_{\mathcal{S}} \leftarrow \phi, \mathcal{C}_{\mathcal{F}} \leftarrow \phi$ ;
2  Initialise the decision variable set $\mathcal{X}$ using the relevant devices in
     $\mathcal{D}$ to the given activities $\mathcal{A}$ ;
3  foreach $A$ in $\mathcal{A}$ do
       // Activity fulfilment constraints.
4      foreach $f$ in $F_A$ do
5          $\mathcal{C}_{\mathcal{A}} \cup (\sum_{\forall d \in \mathcal{D}} X_{A,f,d} = 1)$ ;
6      end
       // Sharing policy constraints.
7      Initialise activity $A$ fulfilment set $\mathcal{I}_A$ ;
8      foreach $P$ in $\mathcal{P}$ do
9          foreach $r$ in $R_P$ do
10             foreach $I_A$ in $\mathcal{I}_A$ do
11                 $\mathcal{C}_{\mathcal{S}} \cup r(I_A)$ ;
12             end
13         end
14     end
15 end
   // Device functions constraints
16 foreach $f \in \cup_{A \in \mathcal{A}} F_A$ do
17     $\mathcal{C}_{\mathcal{F}} \cup (\sum_{\forall A \in \mathcal{A}} X_{A,f,d} \leq Limit(d,f), f \in Cap(d))$ ;
18 end
   // ILP objective function
19 $O = \text{argmax}_{d \in \mathcal{D}} \sum_{X \in \mathcal{X}} Sec(d) \cdot X_{A,d,f}$ ;
20 Return the objective function $O$ and the set of constraints
     $\mathcal{C}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{S}} \cup \mathcal{C}_{\mathcal{F}}$
```

We assume the default sharing policy is to share, as users have trust between them to share their IoT devices [45]. Therefore, it is easier for users to express a few cases where they don't want to share. Another alternative is to set the default policy to *not to share* and any sharing decision should be expressed via sharing policy. This default policy may not be suitable for sharing environment where users share by default as it requires to specify all cases of sharing that instead of the few cases of no sharing. However, it suites the scenarios where users are willing to share in a few situations. In both cases, the final ILP problem solution will tell us which devices to use for which activity function.

5.7.3 ILP Solvers

In the IoTSS system, any ILP problems produced by the ILPPF module will be solved by an off-the-shelf ILP solver such as CP-SAT and CBC solvers [176, 93]. These solvers are expected to identify optimal or near-optimal values for all decision variables $\mathcal{X}$, which defines which device should be used to fulfil each function of user activities. In Section 5.8, we study the efficacy of using these common solvers for a range of IoT device sharing problem instances.

5.7.4 Translation Analysis

After we have defined how to translate the IoT sharing into an ILP problem, we need to analyse the size and complexity of the translated IoT sharing instance. In particular, the number of decision variables and constraints in the corresponding ILP problem. The number of decision variables $|\mathcal{X}|$ is the sum of all possible alternative decisions of fulfilling activities functions which are defined in equation (5.3). The number of ILP constraints is the sum of (a) the total number of functions in all activities $\sum_{A\in\mathcal{A}} |F_A|$, as the fulfilment of each activity function is enforced by a constraint; (b) the total number of device function pairs that are selected to fulfil user activities $|\mathbb{I}_{\mathcal{A}}|$, as each device function requires a constraint to bound its usage below the predefined limit; and (c) the number of user activities multiplied by the total number of rules for all policies, $|\mathcal{A}| \cdot \sum_{p \in \mathcal{P}} |R_p|$. Assuming the most extreme case of rule where each rule constraints the use of a device function to be used in any activity. For example, a rule: don't share device d function f in any activity. This rule needs to be enforced for every activity; hence, it requires a number of constraints equal to the number of activities. In this case, the total number of constraints in the ILP is bounded by Equation (5.9).

$$N_c \leq \sum_{A \in \mathcal{A}} |F_A| + |\mathbb{I}_{\mathcal{A}}| + |\mathcal{A}| \cdot \sum_{p \in \mathcal{P}} |R_p| \tag{5.9}$$

In the other hand, each possible case of fulfilling an activity function is a decision variable. Therefore, the sum of all possible cases of fulfilling all activities is the number of decision variables N_v and this number is calculated by the Equation (5.10).

$$\sum_{A \in \mathcal{A}, d \in \mathcal{D}} |\{(d, f) : f \in F_A, f \in Cap(d)\}| \tag{5.10}$$

5.7.5 IoT Sharing Scenario as ILP Problem

To illustrate how IoT sharing can be translated into an ILP problem, we will use the scenario in Section 5.3. For every user activity, we define all decision variables of fulfilling every function using the available devices. For example, to fulfil activity A's *find* function f_1, there are four available devices, denoted by d_1, d_2, d_3, and d_6. Hence, we define four decision variables $X_{A,d_1,f_1}, X_{A,d_2,f_1}, X_{A,d_3,f_1}$, and X_{A,d_6,f_1} to decide which device to use to fulfil function f_1 of activity A. Only one of these decision variables should have a value of 1. We define the relevant decision variables in Table 5.5 based on decision variables definition in equation (5.3) using the devices information in Table 5.1 and user activities in Table 5.2. These decision variables are used for translating user activity in Table 5.2 by specifying the ILP constraints in this order (a) the constraints that ensure each activity function satisfied by exactly a single device (i.e. $\mathcal{C}_\mathcal{A}$ constraints); (b) the constraints that represent the relevant sharing policies to the user activities (i.e. $\mathcal{C}_\mathcal{S}$); (c) the constraints that ensure that any device's function is not used by a number of activities more than what it can support (i.e. (i.e. $\mathcal{C}_\mathcal{F}$)).

Table 5.5: User activities relevant decision variables.

f_1-**Find**	f_2-**Display**	f_3-**Play**	f_4-**Light**	f_5-**Unlock**	f_6-**Coffee**
X_{A,d_1,f_1} X_{A,d_2,f_1} X_{A,d_3,f_1} X_{A,d_6,f_1}	X_{A,d_1,f_2} X_{A,d_6,f_2}	X_{A,d_1,f_3} X_{A,d_2,f_3} X_{A,d_4,f_3}	X_{A,d_7,f_4}	X_{A,d_5,f_5}	X_{A,d_4,f_6}

Table 5.2 shows that $Jack$'s activity A_1 requires (f_1, f_2, f_4) functions, *Alex* activity A_2 requires (f_5, f_6, f_1, f_3), and user *Guest*'s activity A_3 requires (f_1, f_3). The first set of constraints are defined in (5.11),(5.12), and (5.13), represents the activity fulfilment constraints $\mathcal{C}_\mathcal{A}$ that ensures every activity's function is fulfilled exactly by a single device.

$$\begin{aligned}
&X_{A_1,d_1,f_1} + X_{A_1,d_2,f_1} + X_{A_1,d_3,f_1} + X_{A_1,d_6,f_1} = 1\\
&X_{A_1,d_1,f_2} + X_{A_1,d_6,f_2} = 1\\
&X_{A_1,d_7,f_4} = 1
\end{aligned} \tag{5.11}$$

$$\begin{aligned}
&X_{A_2,d_5,f_5} = 1\\
&X_{A_2,d_4,f_6} = 1\\
&X_{A_2,d_1,f_1} + X_{A_2,d_2,f_1} + X_{A_2,d_3,f_1} + X_{A_2,d_6,f_1} = 1\\
&X_{A_2,d_1,f_3} + X_{A_2,d_2,f_3} + X_{A_2,d_4,f_3} = 1
\end{aligned} \tag{5.12}$$

$$\begin{aligned}
&X_{A_2,d_5,f_5} = 1\\
&X_{A_2,d_4,f_6} = 1\\
&X_{A_2,d_1,f_1} + X_{A_2,d_2,f_1} + X_{A_2,d_3,f_1} + X_{A_2,d_6,f_1} = 1\\
&X_{A_2,d_1,f_3} + X_{A_2,d_2,f_3} + X_{A_2,d_4,f_3} = 1
\end{aligned} \tag{5.13}$$

The second set of constraints represents the user sharing policies (i.e the $\mathcal{C}_{\mathcal{S}}$ constraints), in Table 5.3. Since we assume that the default policy is to share, activity's function can be fulfilled by any device as long as no rule denies it from doing so. Hence, $Jack$'s activity A_1 is not restricted by any policy rule; hence, no constraints for policy enforcement for A_1. In the case of *Alex* activity A_2, *Alex*'s Speaker-rule denies the coffee machine device d_4 to be used with her Speaker d_3. Hence, d_3, d_4 should not be used together for any functions in activity A_2. Note that relevant functions to activity A_4 are f_1, f_3 and f_6, which are reflected by ILP constraints in (5.14). The Guest's activity A_3 is restricted by *Jack*'s TV-rule that denies any user except *Alex* from using his TV. Hence, the TV d_1's find function f_1 and play function f_3 should not be used in A_3. Also, Guest's activity restricted by *Alex's* policy Speaker-rule. Hence, the Speaker d_3's find function f_1 cannot be used the coffee machine d_4's play function f_3 in activity A_3, these two constraints are presented in (5.15).

$$\begin{aligned} &X_{A_2,d_3,f_1} + X_{A_2,d_4,f_3} \leq 1 \\ &X_{A_2,d_3,f_1} + X_{A_2,d_4,f_6} \leq 1 \end{aligned} \tag{5.14}$$

$$\begin{aligned} &X_{A_3,d_1,f_1} = 0 \\ &X_{A_3,d_1,f_3} = 0 \\ &X_{A_3,d_3,f_1} + X_{A_3,d_4,f_3} \leq 1 \end{aligned} \tag{5.15}$$

Finally, across all activities, any device's function should not be used by a number of activities more than its limit, which is enforced by the device functions constraints $\mathcal{C}_{\mathcal{F}}$ in (5.16). Note that if a device's function only required by a single activity, as for functions f_5 and f_6, there is no need to enforce its limit constraints.

$$\begin{aligned}
&X_{A_1,d_1,f_1} + X_{A_2,d_1,f_1} + X_{A_3,d_1,f_1} \leq 10\\
&X_{A_1,d_2,f_1} + X_{A_2,d_2,f_1} + X_{A_3,d_2,f_1} \leq 10\\
&X_{A_1,d_3,f_1} + X_{A_2,d_3,f_1} + X_{A_3,d_3,f_1} \leq 10\\
&X_{A_1,d_6,f_1} + X_{A_2,d_6,f_1} + X_{A_3,d_6,f_1} \leq 10\\
&X_{A_2,d_2,f_3} + X_{A_3,d_2,f_3} \leq 1\\
&X_{A_2,d_4,f_3} + X_{A_3,d_4,f_3} \leq 1
\end{aligned} \tag{5.16}$$

The objective function is formulated similar to our formulation in (5.7) where $\mathcal{X}$ is the set of all decision variables in Table 5.5, the constraints $\mathcal{C}_\mathcal{A}$ are represented by the set of constraints in (5.11),(5.12), (5.13), the constraints $\mathcal{C}_\mathcal{S}$ are represented by (5.14) and (5.15), and the constraints $\mathcal{C}_\mathcal{F}$ are represented by (5.16).

Activities Fulfilment Solution: *Jack*'s activity is not restricted by any sharing rules. Hence, devices with the highest security score will be used for fulfilling his activity, as follows: (1)*Alex*'s Speaker has the highest security score (0.8) from all devices that support find function; (2) Similarly, *Jack*'s TV has the highest score (0.7) from all devices that support display function; (3) Lighting function will be fulfilled by the only device d_7 that support light function. The corresponding decision variables are $(X_{A_1,d_3,f_1}, X_{A_1,d_1,f_2}, X_{A_1,d_7,f_4})$, with total score of $0.8 + 0.7 + 0.65 = 2.15$. Similarly, *Alex* activity will be fulfilled by the available devices that are not violating policy and have the maximum security score, which are: $(X_{A_2,d_5,f_5}, X_{A_2,d_4,f_6}, X_{A_2,d_2,f_1}, X_{A_2,d_2,f_3})$. Note although her Speaker d_3 has the highest security score from the devices that support find function f_1, she cannot use it because her rule doesn't allow it to be used with Coffee machine d_4, which inevitably has to be used for making coffee, see equation (5.14). Finally, the activity for the *Guest* user cannot use *Jack's* TV d_1 and it cannot use *Alex's* Speaker d_3 for find function f_1 with Coffee machine d_4 for play function f_3, see equation (5.15). Therefore, the devices with the highest security score are *Alex's* Speaker for find X_{A_3,d_3,f_1} and

Jack's Speaker for play X_{A_3,d_2,f_3}.

5.8 IoT Secure Sharing Engine Evaluation

We evaluated our sharing policy solving approach by generating artificial IoT sharing network settings and ILP constraints that represent user sharing rules. We evaluated the scalability and the effectiveness of CBC MIP, CP-SAT (both from Google ortools [68]) and the Simulated Annealing (Github project [146]) ILP solvers.

5.8.1 Experimental Setup

To show the feasibility and scalability of solving IoT sharing as an ILP problem, we generated ILP problems that represent IoT sharing instances with varying sizes and complexities. Then we evaluated the performance of multiple commonly used solvers: CBC MIP, CP-SAT, and the Simulated Annealing.

We generate ILP problems as a series of randomly sampled IoT sharing instances with varying sizes and rules based on the network settings shown in Table 5.6. We evaluate the solution score and scalability of solving IoT sharing as optimisation problems using CP-SAT, CBC MIP using Google *ortools* version 7.3.7 [68] both solvers run with the default parameters, and Simulated Annealing (SA) solver using *simanneal* module [146]. SA hyper-parameters has been fine-tuned as in Table 5.7.

We found that these parameters enable SA to perform reasonably well in terms of time and solution score. CP-SAT and CBC MIP default parameters aim to obtain the optimal solution; otherwise, they return "feasible" or "not feasible". In our experiment, we found that all solutions returned by these solvers are optimal solutions. The results reported in the next section are averaged over 30 runs with 95% confidence interval.

ILP problems are generated based on the network setting in Table 5.6

Table 5.6: Network settings for generating the IoT sharing corresponding ILP problems.

Parameter	**Value**
Functions $(n_{\mathcal{F}})$	50
Device function limit $(Limit(d, f))$	random choice {1,10}
Devices $(n_{\mathcal{D}})$	100
Device's security score $(Sec(d))$	random(10,95)
Device capabilities $(n_{Cap(d)})$	5 functions
Activities $(n_{\mathcal{A}})$	10 to 50 activities
Size of activity (n_A)	7-9 functions
Total number of sharing policy rules (n_R)	100 to 1000

Table 5.7: Simulated Annealing settings

Parameter	**Value**
Steps	100*$n_{\mathcal{A}} * n_A$, 10*n_R
Min and Max temperature	0.5, 10,000
Share constraint violation penalty	-500
Requirement constraint violation penalty	-1,500

that guides the size and complexity of the generated problems, in particular, the *search space, decision variables, and constraints*. As Table 5.6, the number of available functions in the network is 50, supported by 100 devices, each of which is capable of performing five random functions. This means on average each function is supported by $n_{df} = n_{\mathcal{D}} * n_{Cap(d)}/n_{\mathcal{F}} = 100 * 5/50 = 10$ devices. Each decision to use device d for function f to support activity A is a binary decision variable. For 10 number of user activities, where each activity requires seven functions; the number of *binary decision variables* is $dv = n_{\mathcal{A}} * n_A * n_{df} = 10 * 7 * 10 = 700$. Consequently, the *search space* of a possible set of devices that support all the ten activities is $2^{dv} = 2^{700}$.

The total number of *constraints* n_R in Table 5.6, as described in (5.9), is the sum of the number of (a) sharing rules constraints n_rs which ranges from 100 to 1,000; (b) activity functions satisfaction constraints which depend on the number of activities and their size. For ten activities each of seven functions $n_A = 10 * 7 = 70$; (c) function limit constraints which depends on the number of devices that support activity functions, the worst case for ten activities $n_{fl} = n_A * n_z * n_{df} = 10*7*10 = 700$. However, practically, the number of function limit constraints is reduced as we don't need to create a rule if the number of activities is less than the function limit. This reduces the number of constraints for ten activities to only $n_{fl} = 165$ constraints.

5.8.2 Results and Discussion

To evaluate the scalability and the effectiveness of the proposed system, we test it under varying numbers of constraints and decision variables. We measure the computation time and the solution security score as well as the number of violated rules if it occurs. We run the experiments in a virtual machine running GNU/Linux Ubuntu 16.04 with five CPUs 3.2GHz and 8GB memory.

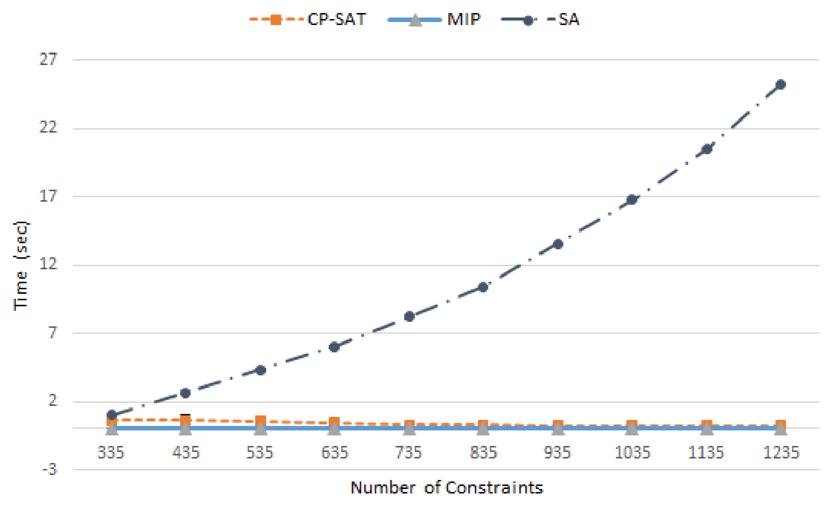


Figure 5.6: Time to find the solution with a different number of constraints.

Number of Constraints

To explore the effect of a number of user policy rules, we vary the number of constraints that represents the sharing rules from 100 to 1,000. These constraints are generated by a uniform random number generator that selects a set of devices and denies them from being used for some functions in user activity. Each constraint represents sharing rule of the form *"don't share device d's function f with user u"*. The total number of constraints is calculated by the Equation (5.9) and the corresponding results are reported in Figures 5.6 and 5.7.

In terms of performance Figure 5.6 shows that the time required by SA to return a solution is exponentially proportional to the number of constraints. This doesn't provide scalability; however, the CBC MIP and CP-SAT solvers both require less than a second regardless of the number of rules. This is because CBC MIP and CP-SAT use the constraints to guide the search. On the other hand, in the case of SA, the time required to validate a given solution is increasing proportionally to the number of constraints.

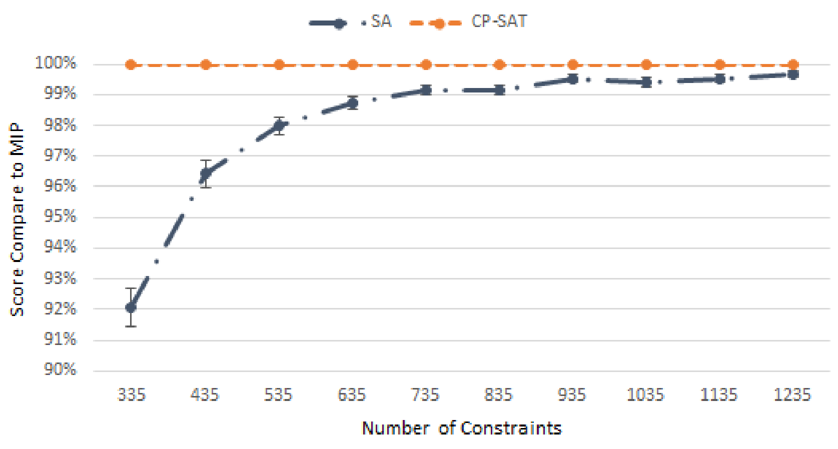


Figure 5.7: Solution score difference ratio to CBC MIP solution score with a different number of constraints for ten activities.

Figure 5.7 shows the solution score as a different ratio to CBC MIP solution score (the maximum score across all solvers). CBC MIP and CP-SAT are guided by an inference propagation that avoids searching on a space that is rejected by the constraints, whereas SA does not support this mechanism. As shown in Figure 5.7, CP-SAT and CBC MIP return same solution score. SA starts with solution 92% of CBC MIP solution score. However, SA solution improves as the number of constraint increase to get the same solution score as CBC MIP when the number of constraints greater than 1,200, but at the expense of time, see Figure 5.6.

In terms of violating constraints, we validate the solutions against the constraints and we found that all solutions satisfied all the constraints. In other words, CBC MIP, CP-SAT and SA return solutions that obey the sharing policy rules, satisfy all activities' requirements, and do not violate any function limit.

Number of Decision Variables

We evaluate CBC MIP, CP-SAT solvers and SA under a different number of decision variables. To control the number of decision variables, we vary

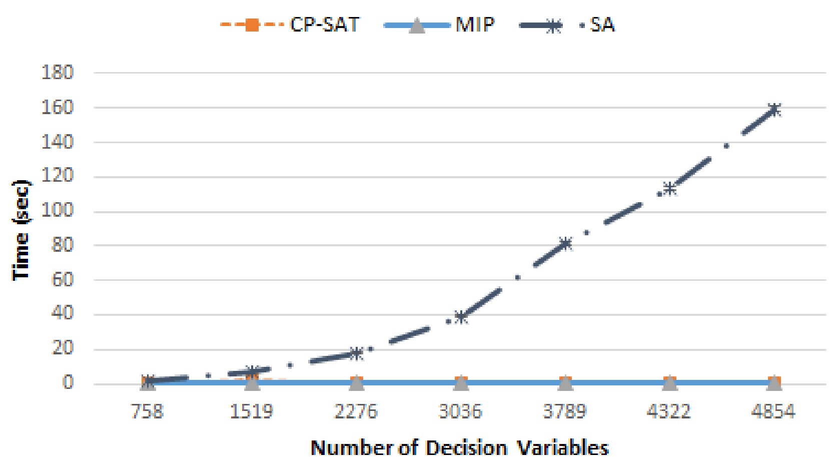


Figure 5.8: Time to find the solution with a different number of decision variables.

the number of activities from 10 to 50 and the number of functions in each activity from 7 to 9. We fixed the number of shared rules constraints to be 500 shared constraints.

Figure 5.8 shows that CP-SAT and CBC MIP solvers outperform SA. The time required by CBC MIP and CP-SAT solvers is less than 0.5 seconds for all problems with 750 up to 4,850 decision variables. Whereas, the time required by SA is increasing exponentially with the number of decision variables. It is noteworthy to mention that 0.5 seconds is a short time compared to the tolerable user waiting time (two seconds [127]).

In terms of the solution score, CBC MIP and CP-SAT return solutions with the same score. Hence, the score difference is zero between CP-SAT and CBC MIP, as shown in Figure 5.9. SA solution score is 88% of CBC MIP and CP-SAT solution score for any all problems with the number of decision variables less than 3,000. However, for problem more than 3,000 constraints, SA solution score significantly decreases to 50% with the increase in the number of decision variables to 4,800.

In summary, SA is inefficient compared to CBC MIP and CP-SAT solvers, especially when the number of decision variables is more than 3,000. More-

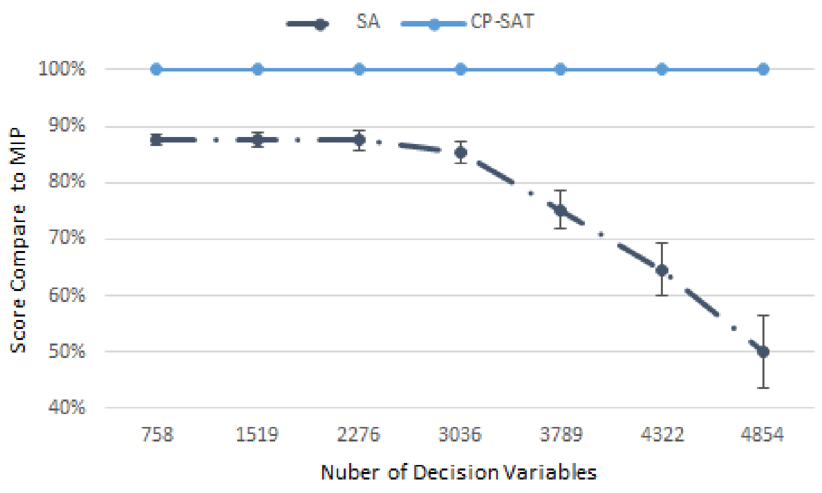


Figure 5.9: Solution score difference ratio to CBC MIP solution score with different decision variables.

over, Figure 5.10 shows that SA starts to violate rule constraints when the number of decision variables increases, even with increasing the constraint violation penalties, see Table 5.7.

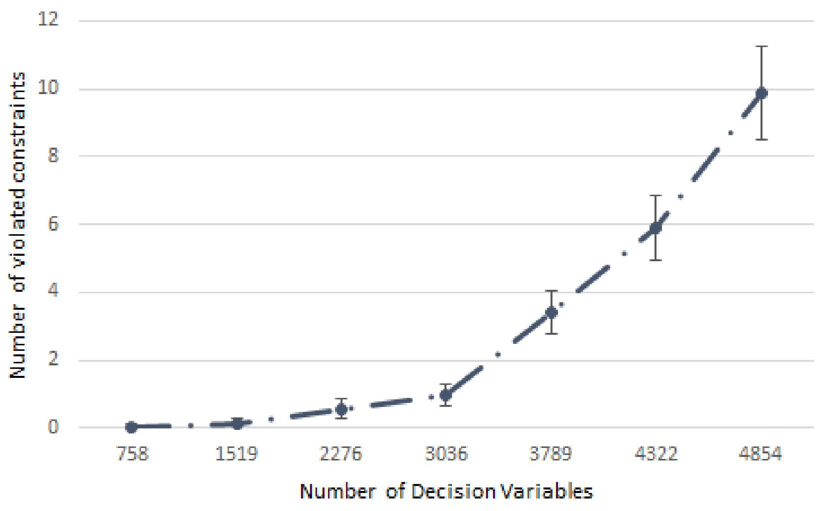


Figure 5.10: SA sharing policy violations.

5.9 Conclusion

This chapter presented a novel approach to solve the smart space IoT sharing problem. We addressed this problem from two aspects: secure device sharing and using secure devices to fulfil user activities. To address the secure sharing aspect, we proposed a newly designed IoT device sharing policy language that enables device owners to clearly define the circumstances under which their devices can be shared. To enable users to use the most secure devices, we proposed to translate IoT device sharing into an ILP problem aiming to minimise the risk of using vulnerable shared devices by maximising the total overall security score of devices selected to fulfil user activities. We mathematically formulated IoT sharing and further transformed it into an equivalent ILP problem. In the ILP formulation, the objective function aims to find the most secure devices to implement user activities, whereas the constraints capture the security aspect. Our experiment results showed that such ILP problems can be solved by the CBC MIP and CP-SAT solvers within 0.5 s for problems with 100 devices, 50 activities, and 1000 policy constraints (note that the tolerable user waiting time is 2 s [127]). In contrast, the simulated annealing solver does not scale well to large problem instances.

Chapter 6

Policy Enforcement Case Study

6.1 Introduction

The previous Chapters (4 and 5) showed how to fulfil user activities and automatically generate access policies automatically. This chapter explores how to implement user activities and enforce their access policies. The main focus is will be on showing the implementation aspect to highlight the practicality of the previous two chapters, particularly especially the PEP component.

We use the WoT technology (see 2.1.2) to implement a smart space case study. WoT enables interoperability in IoT to provide smart services and applications using web protocols. Mozilla Gateway stores and processes user data locally [122] and provides a unified REST API to control the connected devices.

6.1.1 Chapter Goals

This chapter aims to address the following objectives:

- Explore the implementation-related decision for the systems proposed in Chapters 4 and 5.

- Evaluate the PEP (running in the network and application layers) time overhead.

6.1.2 Chapter Organisation

This section uses a smart home, as an example of a typical smart space, to illustrate how user policies, generated in Chapters 4 and 5, can be enforced in the network and the application layers.

6.2 Case Study Overview

6.2.1 Assumptions

This case study is built considering the following assumptions:

1. There is a centralised control point (i.e. hub) that controls all IoT devices using local connections, such as Mozilla Gateway [124].
2. The IoT devices that fulfil user activities and the associated access policies are already selected by the approaches proposed in Chapters 4 and 5.
3. The activities access policies are also provided as an input, and the main focus is on enforcing these policies at the network and application levels.
4. The application-level access control policy assumes that the IoT devices use unencrypted communication, which commonly exists in IoT devices [82].

6.2.2 Threats

Three threats are considered in this experiment to demonstrate the policy enforcement:

- The hub (i.e. device controller) is a valuable target for an adversary, as it can control all devices linked to it. Ideally, the hub should control devices based on direct or automated user activities. However, compromised, vulnerable hubs lead to a direct control of the devices regardless of user activity automation.

- Malicious devices can scan the network and launch local attacks against the IoT devices or the Internet infrastructure.

- Unauthorised use of devices in a shared smart space. For example, the device owner shares the TV for the video watching function, but users with whom the TV is shared can use it for internet browsing.

This case study uses the hub-based architecture, in particular, Mozilla gateway.

6.2.3 IoT Communication Patterns

There are three main communication patterns in IoT framework as follows:

- **Device to cloud**: An IoT device connects to its manufacturer service to update its status and to receive commands. For example, a smart plug connects to and updates its cloud regarding connectivity, status On/Off, and power consumption. It also receives control commands from the cloud to turn On/Off.

- **Hub to device**: Also known as controller to device, communication where a hub locally controls devices that are linked to it without going through their cloud API. Google Home, SmartThings, and AWS IoT Greengrass supports directly control of IoT devices using a LAN connection [67, 170, 12]. This maintains the privacy of users activities private and reduces the unnecessary delay of going through devices' cloud. For example, Google home connects to the security camera and fetches the supported protocols and an authentication token. Then, it forwards this data to the target (e.g. smart TV), and the TV can directly start video streaming without going through Google Home [69].

- **Device to device**: IoT devices can directly connect to a LAN via WiFi or Ethernet. For example, a Raspberry pi can tun on/off a smart plug through WiFi without using the Smart Plug cloud API.

In this case study, we focus on enforcing access control in the network layer at the switch and in the application layer at the IoT proxy such that all IoT device traffic passes through its proxy. This ensures that access control can be enforced in devices' local or external connections.

6.3 Smart Home Case Study

The smart home case study utilises Mozilla Gateway as the centralised hub that controls smart home IoT devices. Mozilla Gateway is a software distribution that implements WoT; hence, it enables users to directly add, link, and control their smart home IoT devices over the web. All devices that are going to be used need to be linked to the Mozilla Gateway before any activity can be built.

In the Mozilla Gateway, a user activity is called a "Rule", which is an "If-then" statement and consists of a condition and an action. The condition is a boolean expression of one or more device's property and the action is a set of one or more device's properties to be changed when the condition is true. For example, "If the door is unlocked, then turn lights on" (see Figure 6.1a). Users activities can be created using API call with the corresponding JSON representation of the activity as shown in Figure 6.1b. In this case study, we build user activities manually using Mozilla Gateway web interface.

Mozilla Gateway evaluates all activities' conditions when the device's properties are changed and keeps device's properties updated using two mechanisms: 1) it pulls all device's property values by default every 5 s; 2) devices send property change messages, as shown in 6.2, when the device's property is changed.

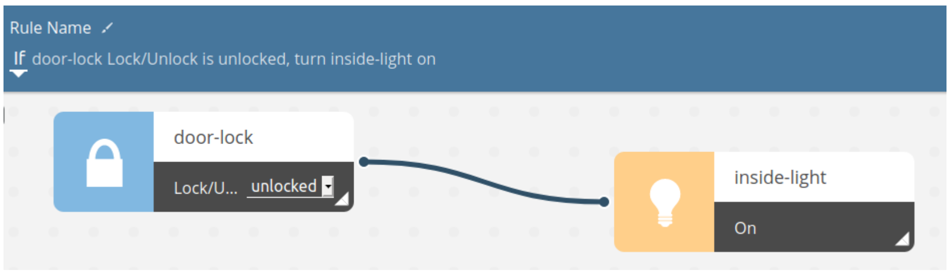
User Activity in Mozilla Gateway

Figure 6.2 illustrates the observed messages that are exchanged in the Mozilla Gateway when an activity, represented as Mozilla Gateway Rule, is triggered. The trigger is used to open the door to trigger the activity. The user interface is a web browser in which a user monitors Mozilla Gateway while the activity is running. In total, there are 15 messages exchanged in the network from the time a door lock open message request is sent to the time the light turns on and users see light on the indicator of the web browser.

In case of door and light activity in Figure 6.1a, *Thing1* represents the door-lock and *Thing2* represents *inside-light*. When the *door-lock* property changes, it sends a property change notification to Mozilla Gateway. Then, Mozilla Gateway checks the activities and verifies that the activity in Figure 6.1a is triggered by *door-lock* property. Mozilla Gateway then sends a change property to turn the light on to *inside-light*.

6.3.1 Device Capabilities and Access Requirements

Device capabilities that are used to automatically fulfil user activities in Chapters 4 and 5 can be extracted from device WoT semantic description. The device WoT semantic description includes the device capabilities and the corresponding URL that needs to be called to call each device capability. For example, Figure 6.3 shows a light bulb semantic as JSON file. The light bulb sends this semantic description to define itself to Mozilla Gateway; hence, it allows it to know which URI to call to use each function.



(a) Mozilla Gateway visual user activity.

```
{
  "enabled": true,
  "trigger": {
    "type": "MultiTrigger",
    "op": "AND",
    "triggers": [
      {
        "type": "EqualityTrigger",
        "label": "Lock/Unlock",
        "property": {
          "type": "string",
          "thing": "http---ec1b506403a0.local-10010",
          "id": "lock",
          "description": "Whether the door is locked or unlocked"
        },
        "value": "unlocked"
      }
    ]
  },
  "effect": {
    "type": "MultiEffect",
    "effects": [
      {
        "type": "ActionEffect",
        "label": "on",
        "thing": "http---3cea87d59eab.local-10025",
        "action": "on",
        "parameters": {}
      }
    ]
  },
  "id": 20,
  "name": "Light on when door unlock"
}
```

(b) Mozilla Gateway POST request body.

Figure 6.1: Mozilla Gateway for implementing user activity “If the door is unlocked, then turn light on”.

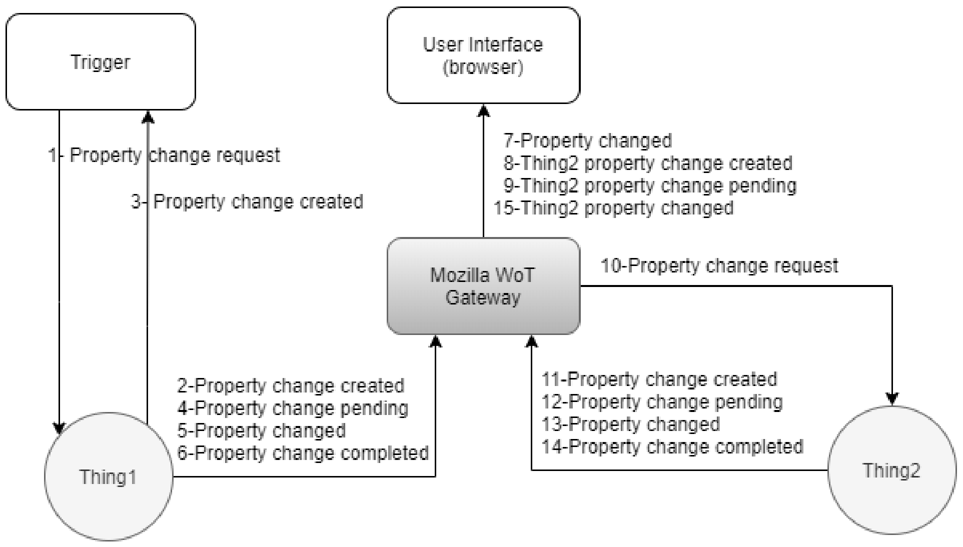
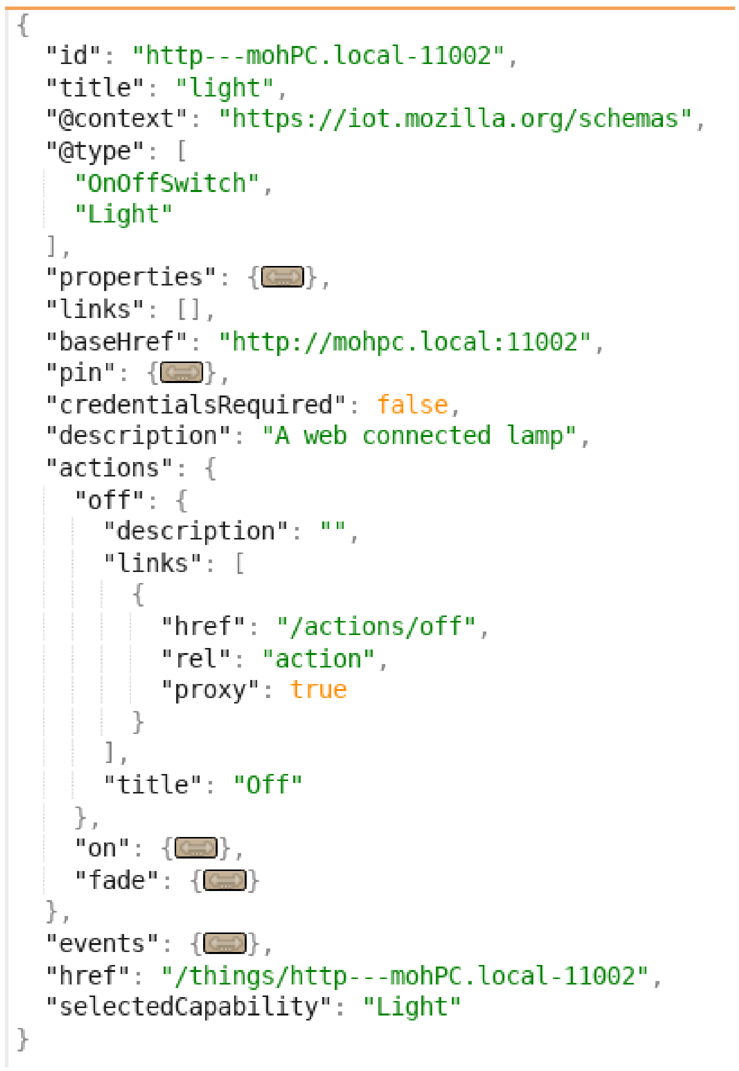

Trigger
User Interface (browser)
1- Property change request
3- Property change created
7-Property changed
8-Thing2 property change created
9-Thing2 property change pending
15-Thing2 property changed
Mozilla WoT Gateway
10-Property change request
2-Property change created
4-Property change pending
5-Property changed
6-Property change completed
11-Property change created
12-Property change pending
13-Property changed
14-Property change completed
Thing1
Thing2

Figure 6.2: Mozilla Gateway messages for activity *"If door opens turn light on"*

This is the feature that we will use to enforce function level access control in this case study.

The capability associated URL is used to build the corresponding access control policy for each activity, as explained in Chapter 5. Mozilla WoT supports a set of IoT device capabilities which includes light, On/Off switch, door sensor capabilities, the list of all supported WoT capability schemas can be found in [123]. Access control in the network layer only allows HTTP and Websocket for this particular case study which use WoT technology.

We assume that the user activities and the corresponding access control are provided as input for this case-study, as shown in Figure 6.4. User activities fulfilment (i.e. concrete activities) can be created by submitting it as a JSON file to Mozilla Gateway via REST API. Access policies are implemented in the switch and in transparent reverse proxies. Each IoT device has its own reverse web proxy that enforces what function can be accessed based on the access control policies (see Figure 6.5). A recent



```
{
  "id": "http---mohPC.local-11002",
  "title": "light",
  "@context": "https://iot.mozilla.org/schemas",
  "@type": [
    "OnOffSwitch",
    "Light"
  ],
  "properties": {…},
  "links": [],
  "baseHref": "http://mohpc.local:11002",
  "pin": {…},
  "credentialsRequired": false,
  "description": "A web connected lamp",
  "actions": {
    "off": {
      "description": "",
      "links": [
        {
          "href": "/actions/off",
          "rel": "action",
          "proxy": true
        }
      ],
      "title": "Off"
    },
    "on": {…},
    "fade": {…}
  },
  "events": {…},
  "href": "/things/http---mohPC.local-11002",
  "selectedCapability": "Light"
}
```

Figure 6.3: Light bulb capabilities declaration.

study showed that smart space still uses unencrypted communication [82, 190] and that around 50% of IoT the traffic is unencrypted [166].

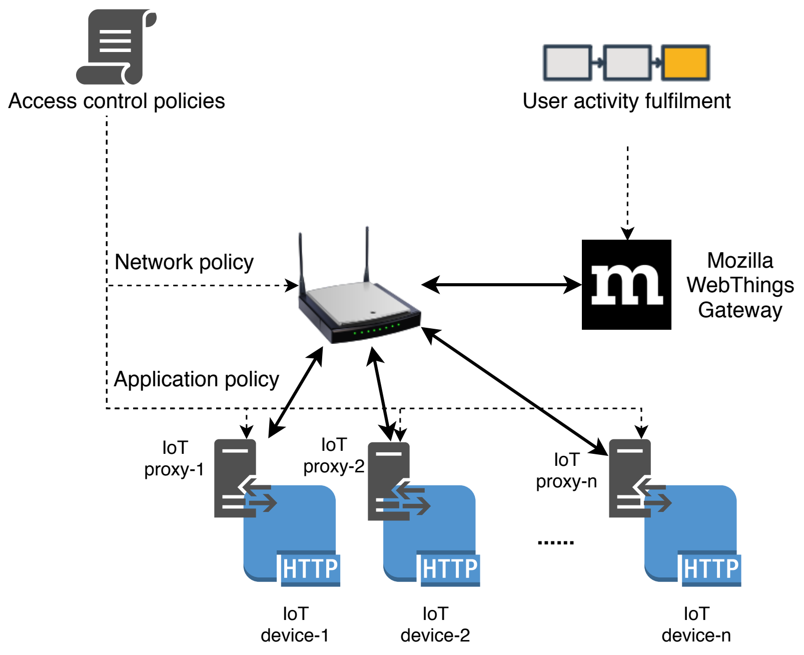


Figure 6.4: Access control implementation in the switch and IoT proxy.

6.4 Experimental Setup

The prototype is an implementation of the diagram shown in Figure 6.4. We use a centralised approach, in which the hub is the central device to control all other devices through a PEP transparent web proxy. The hub unit is the Mozilla Gateway [121], running in a Docker container. IoT devices are created as WoT devices using Python scripts running in a separate Docker container. The experiment runs in a virtual machine running GNU/Linux Ubuntu 16.04 with eight CPUs 3.2GHz and 8GB of memory.

The PEP is implemented in two levels: (a) in the network layer by blocking device-to-device communications, only allowing hub-to-device

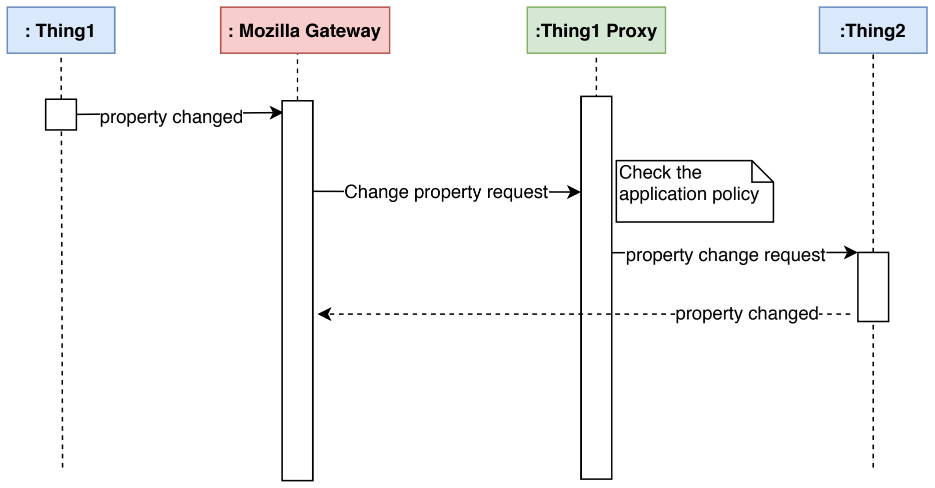


Figure 6.5: PEP as IoT web proxy to enforce application layer access control policies.

communications, as discussed in Section 5.5, and redirecting hub-to-device connections through PEP web proxy; and (b) in the application level in every device proxy (see Figure 6.5), which is implemented as HTTP proxy using node.js HTTP proxy [81]. In this setup, all interactions between IoT devices were local; however, if any external connection is required, then it can be enforced by a proxy that is used for external connections.

PEP proxy intercepts Mozilla Gateway messages to IoT devices to check if the IoT device and the function called is permitted or not using the access policies (see Figure 6.5). For example, given the devices and capabilities in Table 6.2, and the user activities and access policies in Table 6.3, PEP proxy will deny any access to the *Garage* unlock function, as it is not allowed by any policy in 6.3.

Three experiments are conducted to measure the time required to run user activities in different settings with and without PEP. The activity running time is measured to evaluate the effect of PEP on the overall running time. We define the activity running time as the time between the property changed message (message no. 5 in Figure 6.2) that activates the user

Exp.	**Bound IoT devices**	**Activities**	**Parallel**	**Sequential**
1	[5, 10, 15, 20, 25]	2	1	0
2	15	[2, 4, 6, 8, 10]	[1, 2, 3, 4, 5]	0
3	15	[2, 4, 6, 8]	0	[1, 2, 3, 4]

Table 6.1: Experimental settings

activity and the time of completing the action and displaying it to the user (message no. 15 in Figure 6.2). We chose this time because this time is what would be noticeable to a user if there is any delay introduced by the PEP.

Table 6.1 shows the experimental settings of three scenarios based on the following factors:

1. The number of bound devices, which is the number of devices available in the network. This factor is used to control the background traffic to synchronise the IoT devices properties with the Mozilla gateway.

2. The number of parallel activities triggered at a time, in which a single event triggers number of activities at the same time. Hence, this factor controls the number of actions the gateway must execute at once.

3. The sequential number of activities triggered at a time, in which a single event triggers a chain of activities. Hence, this factor controls the number of sequential actions the gateway needs to execute.

A related point to consider is that the number of policy rules is reflected on the number of activities. For example, the activity in Figure 6.1 needs to access the light On/Off function; hence, a single rule is required to allow access to this function (the default policy is to deny).

No	**Device**	**Capabilities**
1	Garage lock	lock, unlock
2	Light	on, off
3	Door lock	lock, unlock
4	Camera outside	stream, take_photo
5	Smart TV	on, off, cast, watch
6	Coffee machine	start, stop
7	Motion sensor	detect
8	Thermostat	off, cooling, heating, auto

Table 6.2: IoT devices types and capabilities

User Activities	**Access Policy**
If Light is off do Garage *lock*	Garage: *lock*
If Garage is unlocked do Light *on*	Light: *on*
If Door is unlocked do Camera *takephoto*	Camera: *takephoto*
If Door is locked do Camera *takephoto*	Camera: *takephoto*
If Smart TV is in watch do Coffee machine *start*	Coffee machine: *start*
If Garage is unlocked do Coffee machine *start*	Coffee machine: *start*
If Motion sensor is detected do Thermostat *auto*	Thermostat: *auto*
If Smart TV is off do Thermostat *auto*	Thermostat: *auto*

Table 6.3: Examples of user activities and the corresponding allowed access capabilities.

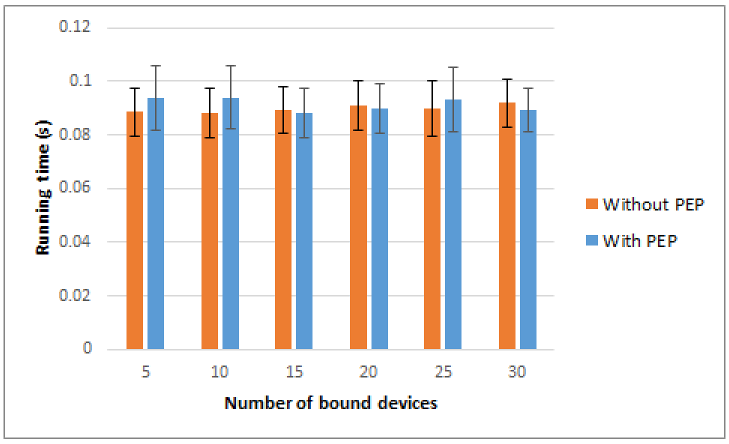


Figure 6.6: Trigger to action time with a different number of bound devices.

6.5 Results and Discussion

The experimental results of PEP performance show that the average running time of five concurrent activities is 0.255 ± 0.024 s, from which 0.008 s is the time overhead introduced by PEP. We performed three experiments, as described in Table 6.1. Each experiment was executed 36 times.

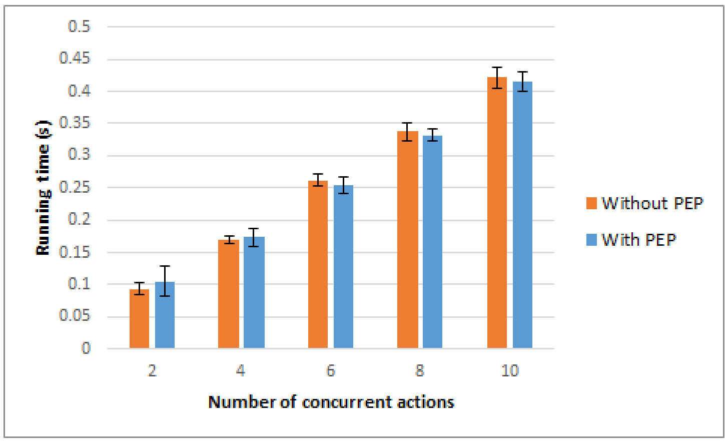


Figure 6.7: Trigger to action time with a different number of actions at a time.

The results from the first experiment, in Figure 6.6, show the impact of the number of devices linked to the gateway on the required time to execute user activity. As depicted in Figure 6.6, when the number of linked IoT devices is up to 25, the background device's property values synchronisation has no significant effect on the activity running time. The average running time of an activity is around 82 ms. It also shows that the PEP time overhead is as small as the margin error of 95% confidence interval.

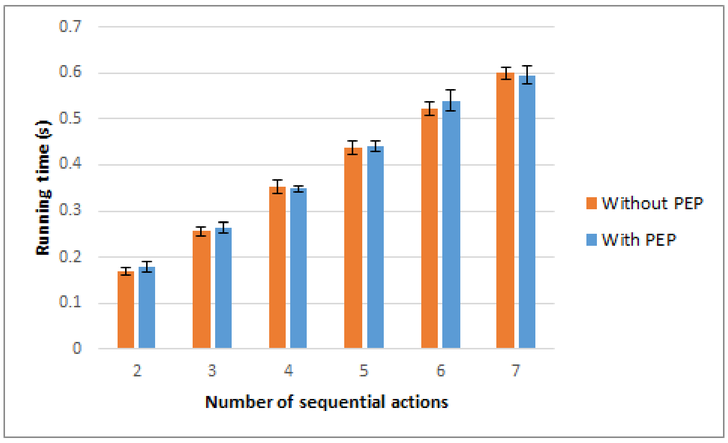


Figure 6.8: Trigger to action time with a different number of devices.

The results of the second experiment show that the activity running time increases proportionally to the number of concurrent activated actions (see Figure 6.7). However, as in the previous results, the time overhead introduced by PEP is not statistically significant.

The results from the third experiment, in Figure 6.8, show the impact of running sequential activities. The time to execute all activities increases proportionally to the number of sequential activities. This is because the gateway has to execute all functions in the activities in sequential order. The running time for sequential activities is significantly higher than concurrent activities, as shown in Figures 6.8 and 6.7. However, as in the previous results, the time overhead introduced by PEP is negligible.

6.6 Conclusion

The case study results indicated that runtime policy enforcement can be achieved by adopting a transparent proxy service to be jointly used with the Mozilla Gateway. Evaluation experiments showed that enforcing fine-grained policies in the network and application layers do not introduce a statistically significant overhead on the running activities.

Chapter 7

Conclusions and Future Work

This chapter concludes the discussion of this thesis, highlights the main findings, and outlines directions for future works.

The overall goal of this thesis was to develop a new method to use user input to derive fine-grained access policies. This goal was successfully achieved by developing a security framework, and the innovative formulation of user activity fulfilment as optimisation problems that use user preference and sharing policies to automatically fulfil user activities. We developed algorithms to derive network and application fine-grained access control policies for the devices that are selected to fulfil user activities. The validation test of the newly proposed security framework showed its usefulness and practicality in mitigating unauthorised network access to vulnerable IoT devices in the smart space. The results of formulating activity fulfilment as an optimisation problem using user preferences show that GA efficiently finds the solution. In addition, results indicate that by formulating IoT sharing as an ILP problem, off-the-shelf ILP solvers can efficiently find the most secure set of devices that fulfil user activities without violating any sharing policy.

The remainder of this chapter is organised as follows: Section 7.1 provides the individual objectives that were achieved in this thesis. Section 7.2highlights the main thesis conclusions and findings. Discussions and suggestions for several potential research directions are presented in Section 7.3.

7.1 Achieved Objectives

This thesis has successfully fulfilled the following research objectives:

- This thesis developed a security framework (Chapter 3) which enables users to customise pre-defined IoT policies to fulfil their requirements. The framework also integrates network security middleboxes to mitigate security attacks. Two security services have been developed in the framework: IPv4 ARP spoofing server to mitigate IPv4 ARP spoofing attacks and IDS integration to mitigate ongoing attacks and automatically block malicious traffic. The framework validation test shows that user input reduces the attack surface on IoT devices while blocking connections that are not used by any user activity. Moreover, the study of IPv4 ARP spoofing server shows that it can effectively prevent ARP spoofing attacks. By bypassing the kernel, the ARP server outperforms existing SDN centralised ARP servers.

- This thesis proposed a new intelligent approach (Chapter 4) that represents user activities as functional workflows to be used to automatically fulfil user activities using the preferred set of devices. The problem was formulated as a constraint optimisation task, which was solved using heuristic searching algorithms to automatically select the set of devices that satisfy the functional workflow requirements and maximise user preferences. A policy generation algorithm was proposed to systematically generate network access policies using the network requirements for each device function to enforce the principle of least privilege on the selected devices. The experiment results showed that the GA outperforms the other searching algorithms in terms of time and quality of the solution.

- This thesis proposed a novel method (Chapter 5) to automatically fulfil user activity functions using the most secure set of devices without violating user sharing policies in a shared smart space. To the best of our knowledge, we are the first to formulate this as an ILP problem. The formulation relies on translating sharing policies into ILP constraints and quantifying device security to be maximised using the objective function. An algorithm was developed to automatically transform an IoT sharing problem into an ILP problem. The experimental results showed that the proposed method is effective at finding the most secure devices to fulfil user activities without violating user sharing policies.

- This thesis presented an access control case study (Chapter 6) to evaluate the feasibility and practicality of using the proposed fine-grained access policies framework using existing technologies. User activities and the corresponding network and application policies, presented in Chapters 4 and 5, were provided as inputs to the case study to demonstrate how it can be enforced using existing hub-based smart spaces, such as Mozilla Gateway. The results of experiments conducted in the case study showed that network and application policy enforcement can be integrated into existing IoT hub-based access control systems without incurring a time overhead noticeable to users.

7.2 Main Conclusions

Overall, this thesis explored the possibility of building fine-grained access control through utilising (a) user manual input to customise pre-defined IoT policies to meet their security and privacy requirements and (b) user preferences and sharing policies to automatically derive activity fulfilment and the corresponding access control policies automatic.

Chapters 3, 4, and 5 propose methods to use different types of user inputs to customise and derive access control policies. Results from Chapters 3 and 4 show that fine-grained access control requires some user inputs that can define which devices are used by user activities and how to derive fine-grained access control assuming. Chapter 6 results shows that the proposed access control system can be implemented using commonly used IoT hub such as Mozilla Gateway.

The main conclusions drawn from each of the four contribution chapters (Chapters 3 to 6) are presented and discussed in this section.

7.2.1 User-Centric Access Control Framework

Chapter 3 proposes an access control framework that enables users to tailor IoT device policies to meet their security and privacy requirements. The IoT device policies are considered as the minimum security policy that can be restored to cover any user mistakes. Moreover, the framework integrates IDS to mitigate malicious activities within the allowed connections and block them using SDN. In addition, a centralised ARP server has been implemented as security services in the framework. Validation results demonstrate the effectiveness of integrating user access rules into existing security countermeasures (i.e. pre-defined policies and IDS) to enforce user security and privacy. The results also indicate that implementing security applications such as ARP server in the data plane can easily support smart home network traffic.

7.2.2 Optimising User Preference for Fine-grained Access Control

This thesis presents a new important problem, which is how to use user preferences to derive fine-grained access control. Solving this problem can help to consider user access control decisions in a transparent manner. To solve this problem, we had to initially identify which devices should be used to fulfil user activities, and then derive access policies for the selected devices. We formulated the problem of device selection as an optimisation problem to find the set of devices that fulfil user activity while optimising user preference (Chapter 4). Based on this formulation, local and global optimisation search algorithms were used. Searching algorithms were guided by a user preference quantified model that returns a scalar value for each given set of devices. An algorithm has been proposed to generate an access control policy for each set of devices to support user activities. The results indicated that global optimisation search algorithms, such as GA, are the most efficient and effective compared to local search algorithms, like such as simulated annealing and hill-climbing.

The results help us to understand the performance of optimisation search algorithms in terms of efficiency and scalability in automatically fulfilling user activities. Moreover, the problem formulation identified a new manner of using user input to derive access control that reflects user preferences.

7.2.3 Optimisation for Secure Sharing and Secure Using

Chapter 5 identifies a new problem in a multi-user smart space, called IoT sharing problem. The IoT sharing problem is how to enable users to share and use their IoT devices securely. This problem has been addressed by proposing a new sharing language to enable users to specify their sharing conditions. A novel formulation has been proposed to translate the IoT sharing problem into an ILP problem to fulfil user activities using the most secure set of devices without violating user sharing policies. The formulation converts user sharing policies into a set of constraints and maximises quantified device security in the objective function. Results showed that the IoT sharing problem can naturally be translated into an ILP problem and effectively use off-the-shelf ILP solvers.

This thesis introduces a new approach of using optimisation techniques for addressing security problems in a multi-user smart space. It also helps us to understand the efficiency and scalability of ILP solvers when they are used to solve the IoT sharing problem.

7.2.4 Fine-grained Policy Enforcement

This thesis proposes network and application policies that mitigate the threat of exploiting vulnerable IoT devices by restricting network and application communications to the minimum necessary to support user activities. Moreover, device function abuse can be effectively mitigated by application policies that limit connected devices to using only the functions that are part of user activities. From the case study in Chapter 6, it has been found that policy enforcement can be achieved by adopting a device proxy service for each IoT device. In addition, access control based on device functions is suitable to be used with WoT technology. The WoT provides a semantic description that includes what functions/capabilities a device (i.e. Thing) supports with the associated URIs to invoke them. Therefore, access control policies can be constructed using the semantic descriptions of the IoT devices. Evaluation experiments show that network and application policies can be enforced without incurring any significant overhead on the user activity running time.

7.2.5 Function-based Access Control Policy

Chapter 4 and 5 use functional workflows to represents user activity functional requirements to derive fine-grained access policies based on what functions are required by user activities and the device function access requirements. It has been found that abstracting user activities using their functions is useful to systematically derive fine-grained access control policies. Consequently, these policies enforce the least privilege access control.

Furthermore, decoupling user activities from the underlying devices can enable users to reuse pre-defined user activities, which can be shared in a repository.

7.3 Future Work

Finally, this section provides possible research directions for future work.

7.3.1 Access Control Quality of Experience aware

This thesis proposes the first approach to automatically generate access control policies for static user activities (Chapter 4) by leveraging functional workflow representation for user activities. It has been found that decoupling user activities from the underlying devices open opportunities to derive access control policies and meet user requirements (e.g. user preferences). However, user activities can dynamically change due to many factors such as user mobility and QoE [13, 16, 32]. For example, video content streaming function, in a user activity, needs to be fulfilled by the closest device to a user to improve user QoE. However, enforcing access policy on user activities may affect their QoE, particularly with dynamically changed activities. The challenge is how to support dynamic fulfilment and access control without affecting user QoE.

User experience also includes resolving the conflict when more than one user activity requires access to a device function. Such conflict can be solved as a scheduling problem with priority based on the users (owners first) or the activity task (studying activities have higher priority than entertainment ones).

7.3.2 User Activities with Web Services

This thesis addressed mainly the use of user input to derive access control for IoT devices; however, the proposed methods can also support external web services. User activities, represented as functional workflows, may have some functions that IoT devices can fulfil, whereas others need external services. Similarly to the local network requirements of IoT devices, access control rules to external services are required to support activities that need to access web services. The user preference model needs to include the devices and functions preferred by users and also which services for each function. In a sharing scenario, web services do not affect how users share their devices, except if they do not want to share them. Thus, the sharing policy language can be extended to consider web services.

7.3.3 Cross Smart Spaces Device Sharing

IoT sharing often refers to users sharing IoT devices in the same smart space. However, IoT sharing can happen across smart spaces when user activities need to be fulfilled by devices located in different spaces. For example, user A may build an activity to turn the light red when the smoke sensor goes off in user B home. Although this can be considered as external services, managing users from different smart spaces to provide a wider sharing environment needs further investigation.

7.3.4 Statefull IoT Sharing

This thesis considered IoT devices as stateless in IoT sharing problem and focused on the sharing policies and access control. However, when devices are shared between multiple users, an operational conflict may happen. The conflict in the operation can be solved, for example, by prioritising user activities. However, further research that consider IoT devices state into network access control is required.

7.3.5 Multi-Objective Device Selection Using User Preferences and Device Security

This thesis found that the optimisation approach efficiently solves smart space problems with a single objective (i.e. user preference, device security). However, if the goal is to fulfil user activities using the user preferred device and the most secure device, then these two objectives might be conflicting. For example, a user may prefer to use a device that is less secure than another. Future work could investigate the multi-objective optimisation technique to meet user preferences and increase the security of the smart space by using secure devices when available. Simultaneously, optimising several objectives is a challenging task, requiring further studies on existing multi-objective optimisation techniques that suit this scenario.

Bibliography

[1] Acar, G., Huang, D. Y., Li, F., Narayanan, A., and Feamster, N. Web-based attacks to discover and control local iot devices. In *Proceedings of the 2018 Workshop on IoT Security and Privacy* (2018), pp. 29–35.

[2] Al-Shaboti, M. Smart home access control system. https://www.youtube.com/watch?v=T9a0UI85cYA (Accessed on 15/12/2019).

[3] Al-shaboti, M. Dam, 2020. Last accessed 5/1/2020.

[4] Al-Shaboti, M., Aaron, C., and Ian, W. Automatic Device Selection and Access Policy Generation based on User Preference for IoT Activity Workflow. In *eprint arXiv:1904.06495* (Apr. 2019).

[5] Al-Shaboti, M., Welch, I., Chen, A., and Mahmood, M. A. Towards secure smart home iot: Manufacturer and user network access control framework. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)* (2018), IEEE, pp. 892–899.

[6] Alaa, M., Zaidan, A., Zaidan, B., Talal, M., and Kiah, M. A review of smart home applications based on internet of things. *Journal of Network and Computer Applications 97* (2017), 48–65.

[7] ALI, S. T., SIVARAMAN, V., RADFORD, A., AND JHA, S. A survey of securing networks using software defined networking. *IEEE transactions on reliability 64*, 3 (2015), 1086–1097.

[8] ALKHRESHEH, A., ELGAZZAR, K., AND HASSANEIN, H. S. Context-aware automatic access policy specification for iot environments. In *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)* (2018), IEEE, pp. 793–799.

[9] AMAN, W., AND SNEKKENES, E. Event driven adaptive security in internet of things. In *Proceedings of the Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2014), Rome, Italy* (2014), vol. 2428, p. 715.

[10] AMAZON. Echo (3rd gen) - smart speaker with alexa, 2020. Last accessed 15/2/2020.

[11] ANDREASSON, N., PATRIKSSON, M., AND EVGRAFOV, A. *An introduction to continuous optimization: foundations and fundamental algorithms*. Courier Dover Publications, 2020.

[12] AWS. Aws iot greengrass. https://aws.amazon.com/greengrass/ (Accessed on 27/5/2020).

[13] BAEK, K.-D., AND KO, I.-Y. Spatially cohesive service discovery and dynamic service handover for distributed iot environments. In *International Conference on Web Engineering* (2017), Springer, pp. 60–78.

[14] BAJPAI, P., AND KUMAR, M. Genetic algorithm–an approach to solve global optimization problems. *Indian Journal of computer science and engineering 1*, 3 (2010), 199–206.

[15] BAKKER, J. N., WELCH, I., AND SEAH, W. K. Network-wide virtual firewall using sdn/openflow. In *Network Function Virtualization*

and Software Defined Networks (NFV-SDN), IEEE Conference on (2016), IEEE, pp. 62–68.

[16] BAO, W., YUAN, D., YANG, Z., WANG, S., LI, W., ZHOU, B. B., AND ZOMAYA, A. Y. Follow me fog: toward seamless handover timing schemes in a fog computing environment. *IEEE Communications Magazine 55*, 11 (2017), 72–78.

[17] BARRERA, D., MOLLOY, I., AND HUANG, H. Standardizing iot network security policy enforcement. In *Workshop on Decentralized IoT Security and Standards (DISS)* (2018), vol. 2018, p. 6.

[18] BELLAVISTA, P., AND MONTANARI, A. Context awareness for adaptive access control management in iot environments. *Secur. Priv. Cyber-Phys. Syst.: Found. Princ. Appl 2*, 5 (2017), 157–178.

[19] BERGIUS, H. Desktop summit, and some thoughts on flow-based programming, 2010. Last accessed 10/7/2020.

[20] BHUNIA, S. S., AND GURUSAMY, M. Dynamic attack detection and mitigation in iot using sdn. In *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)* (2017), IEEE, pp. 1–6.

[21] BLACKSTOCK, M., AND LEA, R. Fred: a hosted data flow platform for the iot built using node-red. *Proceedings of MoTA* (2016).

[22] BREWER, D. F., AND NASH, M. J. The chinese wall security policy. In *null* (1989), IEEE, p. 206.

[23] BUNTZ, B. Sound data privacy policy can drive market differentiation, 2020. Last accessed 7/5/2020.

[24] CASADO, M., FREEDMAN, M. J., PETTIT, J., LUO, J., MCKEOWN, N., AND SHENKER, S. Ethane: Taking control of the enterprise.

In *ACM SIGCOMM Computer Communication Review* (2007), vol. 37, ACM, pp. 1–12.

[25] CASADO, M., GARFINKEL, T., AKELLA, A., FREEDMAN, M. J., BONEH, D., MCKEOWN, N., AND SHENKER, S. Sane: A protection architecture for enterprise networks. In *Usenix Security* (2006).

[26] CECCHINATO, M., AND HARRISON, D. Degrees of agency in owners and users of home iot devices. In *CHI'17 workshop: Making Home: Asserting Agency in the Age of IoT* (2017), Association for Computing Machinery (ACM).

[27] CELIK, Z. B., MCDANIEL, P., AND TAN, G. Soteria: Automated iot safety and security analysis. In *2018 {USENIX} Annual Technical Conference ({$USENIX$}{ATC} 18)* (2018), pp. 147–158.

[28] CELIK, Z. B., TAN, G., AND MCDANIEL, P. D. Iotguard: Dynamic enforcement of security and safety policy in commodity iot. In *NDSS* (2019).

[29] CHEN, J., ZUO, C., DIAO, W., DONG, S., ZHAO, Q., SUN, M., LIN, Z., ZHANG, Y., AND ZHANG, K. Your iots are (not) mine: On the remote binding between iot devices and users. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2019), IEEE, pp. 222–233.

[30] CHETTY, M., KIM, H., SUNDARESAN, S., BURNETT, S., FEAMSTER, N., AND EDWARDS, W. K. ucap: An internet data management tool for the home. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (2015), ACM, pp. 3093–3102.

[31] CHO, H., KANG, S., AND LEE, Y. Centralized arp proxy server over sdn controller to cut down arp broadcast in large-scale data center networks. In *2015 International Conference on Information Networking (ICOIN)* (2015), IEEE, pp. 301–306.

[32] CHO, J.-H., KO, H.-G., AND KO, I.-Y. Adaptive service selection according to the service density in multiple qos aspects. *IEEE Transactions on Services Computing 9*, 6 (2016), 883–894.

[33] CLOUDFLARE. Inside the infamous mirai iot botnet: A retrospective analysis, Dec. 2017.

[34] CLOUDFLARE. Reverse proxy, 2020. Last accessed 25/7/2020.

[35] CONFORTI, M., CORNUÉJOLS, G., ZAMBELLI, G., ET AL. *Integer programming*, vol. 271. Springer, 2014.

[36] CONSTANTIN, L. Hackers found 47 new vulnerabilities in 23 iot devices at def con, Sept. 2016.

[37] COX, J. H., CLARK, R. J., AND OWEN, H. L. Leveraging sdn for arp security. In *SoutheastCon, 2016* (2016), IEEE, pp. 1–8.

[38] CRANOR, L. F. A framework for reasoning about the human in the loop. *UPSEC 8*, 2008 (2008), 1–15.

[39] DALY, R., SHEN, Q., AND AITKEN, S. Learning bayesian networks: approaches and issues. *The knowledge engineering review 26*, 2 (2011), 99–157.

[40] DEMARINIS, N., AND FONSECA, R. Toward usable network traffic policies for iot devices in consumer networks. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy* (2017), pp. 43–48.

[41] DICKSON, B. More smart home devices vulnerable, mcafee researchers find, 2020. Last accessed 15/7/2020.

[42] DING, A. Y., CROWCROFT, J., TARKOMA, S., AND FLINCK, H. Software defined networking for security enhancement in wireless mobile networks. *Computer Networks 66* (2014), 94–101.

[43] DING, W., AND HU, H. On the safety of iot device physical interaction control. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), ACM, pp. 832–846.

[44] EDWARDS, W. K., POOLE, E. S., AND STOLL, J. Security automation considered harmful? In *Proceedings of the 2007 Workshop on New Security Paradigms* (2008), ACM, pp. 33–42.

[45] ENCK, W., ONGTANG, M., AND MCDANIEL, P. Understanding android security. *IEEE security & privacy 7*, 1 (2009), 50–57.

[46] ENTERPRISE, H. Internet of things research study. Tech. rep., HP, 2015.

[47] FEAMSTER, N., REXFORD, J., AND ZEGURA, E. The road to sdn. *Queue 11*, 12 (2013), 20.

[48] FERNANDES, E., JUNG, J., AND PRAKASH, A. Security analysis of emerging smart home applications. In *2016 IEEE Symposium on Security and Privacy (SP)* (2016), IEEE, pp. 636–654.

[49] FERRAG, M. A., MAGLARAS, L. A., JANICKE, H., JIANG, J., AND SHU, L. Authentication protocols for internet of things: a comprehensive survey. *Security and Communication Networks 2017* (2017).

[50] FETH, D., MAIER, A., AND POLST, S. A user-centered model for usable security and privacy. In *International Conference on Human Aspects of Information Security, Privacy, and Trust* (2017), Springer, pp. 74–89.

[51] FLECHAIS, I., RIEGELSBERGER, J., AND SASSE, M. A. Divide and conquer: the role of trust and assurance in the design of secure sociotechnical systems. In *Proceedings of the 2005 workshop on New security paradigms* (2005), ACM, pp. 33–41.

[52] FLOW TEAM, M. Microsoft flow. `https://flow.microsoft.com/en-us/` (visited on 18/10/2018).

[53] FURINI, M., MANDREOLI, F., MARTOGLIA, R., AND MONTANGERO, M. Iot: Science fiction or real revolution? In *International Conference on Smart Objects and Technologies for Social Good* (2016), Springer, pp. 96–105.

[54] FÜRNKRANZ, J., AND HÜLLERMEIER, E. Preference learning and ranking by pairwise comparison. In *Preference learning*. Springer, 2010, pp. 65–82.

[55] GAGLIARDI, V. Introduction to the decoupled world. In *Decoupled Django*. Springer, 2021, pp. 1–15.

[56] GANJI, A., PAGE, G., AND SHAHZAD, M. Characterizing the performance of wifi in dense iot deployments. In *2019 28th International Conference on Computer Communication and Networks (ICCCN)* (2019), IEEE, pp. 1–9.

[57] GARBIS, J. The software defined perimeter, 2017. [Online; accessed 15-April-2017].

[58] GARG, R., AND MORENO, C. Understanding motivators, constraints, and practices of sharing internet of things. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 3*, 2 (2019), 44.

[59] GARTNER. Gartner says 5.8 billion enterprise and automotive iot endpoints will be in use in 2020. https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io (Accessed on 15/1/2020).

[60] GARTNER. Gartner says a typical family home could contain more than 500 smart devices by 2022, 2014. https://www.gartner.com/en/newsroom/press-releases/2014-09-08-gartner-says-a-typical-family-home-could-contain-more-than-500-smart-devices-by-2022 (visited on 8/4/2019).

[61] GE, M., HONG, J. B., GUTTMANN, W., AND KIM, D. S. A framework for automating security analysis of the internet of things. *Journal of Network and Computer Applications 83* (2017), 12–27.

[62] GEENG, C., AND ROESNER, F. Who's in control?: Interactions in multi-user smart homes. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (2019), ACM, p. 268.

[63] GHADYANI, K. Networking 215+ smart home devices, 2020. Last accessed 24/6/2020.

[64] GHARAKHEILI, H. H., SIVARAMAN, V., MOORS, T., VISHWANATH, A., MATTHEWS, J., AND RUSSELL, C. Enabling fast and slow lanes for content providers using software defined networking. *IEEE/ACM Transactions on Networking* (2016).

[65] GIANG, N. K., BLACKSTOCK, M., LEA, R., AND LEUNG, V. C. Developing iot applications in the fog: a distributed dataflow approach. In *Internet of Things (IOT), 2015 5th International Conference on the* (2015), IEEE, pp. 155–162.

[66] GOOGLE. Local fulfillment. https://developers.google.com/assistant/smarthome/conce traits (Accessed on 15/4/2020).

[67] GOOGLE. Local fulfillment. https://developers.google.com/assistant/smarthome/conce (Accessed on 15/4/2020).

[68] GOOGLE. Or-tools. https://developers.google.com/optimization (Accessed on 15/1/2020).

[69] GOOGLE. Smart home camerastream trait schema. https://developers.google.com/assistant/smarthome/traits/camerastream (Accessed on 15/4/2020).

[70] GOUTAM, S., ENCK, W., AND REAVES, B. Hestia: simple least privilege network policies for smart homes. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks* (2019), pp. 215–220.

[71] GROSSMAN, D., AND DOMINGOS, P. Learning bayesian network classifiers by maximizing conditional likelihood. In *Proceedings of the twenty-first international conference on Machine learning* (2004), ACM, p. 46.

[72] HAMZA, A., GHARAKHEILI, H. H., AND SIVARAMAN, V. Combining mud policies with sdn for iot intrusion detection. In *Proceedings of the 2018 Workshop on IoT Security and Privacy* (2018), ACM, pp. 1–7.

[73] HAMZA, A., RANATHUNGA, D., GHARAKHEILI, H. H., ROUGHAN, M., AND SIVARAMAN, V. Clear as mud: Generating, validating and applying iot behavioral profiles. In *Proceedings of the 2018 Workshop on IoT Security and Privacy* (2018), ACM, pp. 8–14.

[74] HAMZEI, M., AND NAVIMIPOUR, N. J. Toward efficient service composition techniques in the internet of things. *IEEE Internet of Things Journal 5*, 5 (2018), 3774–3787.

[75] HANEY, J. M., FURMAN, S. M., AND ACAR, Y. Smart home security and privacy mitigations: Consumer perceptions, practices, and challenges. In *International Conference on Human-Computer Interaction* (2020), Springer, pp. 393–411.

[76] HARDT, D. The oauth 2.0 authorization framework. RFC 6749, RFC Editor, October 2012. `http://www.rfc-editor.org/rfc/rfc6749.txt`.

[77] HARRISON, R., CAI, Q., GUPTA, A., AND REXFORD, J. Network-wide heavy hitter detection with commodity switches. In *Proceedings of the Symposium on SDN Research* (2018), pp. 1–7.

[78] HASSAN, W. H., ET AL. Current research on internet of things (iot) security: A survey. *Computer networks 148* (2019), 283–294.

[79] HAYES, M., NG, B., PEKAR, A., AND SEAH, W. K. Scalable architecture for sdn traffic classification. *IEEE Systems Journal* (2017).

[80] HE, W., GOLLA, M., PADHI, R., OFEK, J., DÜRMUTH, M., FERNANDES, E., AND UR, B. Rethinking access control and authentication for the home internet of things (iot). In *27th {USENIX} Security Symposium ({USENIX} Security 18)* (2018), pp. 255–272.

[81] HTTP PARTY. Nodejs http proxy, 2020. `https://github.com/http-party/node-http-proxy` (accessed on 2/5/2020).

[82] HUANG, D. Y., APTHORPE, N., LI, F., ACAR, G., AND FEAMSTER, N. Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 4*, 2 (2020), 1–21.

[83] IFTTT. Ifttt, 2020. Last accessed 5/7/2020.

[84] INC., S. Smartthings. https://www.smartthings.com/ (Accessed on 5/2/2020).

[85] INITIATIVE, I. I., ET AL. Towards a definition of the internet of things (iot). *Revision-1, on-line: http://iot. ieee. org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15. pdf. Accessed 27*, 2017 (2015), 479–501.

[86] JACOBSSON, A., BOLDT, M., AND CARLSSON, B. A risk analysis of a smart home automation system. *Future Generation Computer Systems 56* (2016), 719–733.

[87] JAFARIAN, J. H., AL-SHAER, E., AND DUAN, Q. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks* (2012), ACM, pp. 127–132.

[88] JAIN, A., SINGH, T., AND SHARMA, S. K. Threats paradigmin iot ecosystem. In *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)* (2018), IEEE, pp. 1–7.

[89] JAIN, S., KUMAR, A., MANDAL, S., ONG, J., POUTIEVSKI, L., SINGH, A., VENKATA, S., WANDERER, J., ZHOU, J., ZHU, M., ET AL. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review 43*, 4 (2013), 3–14.

[90] JANG, W., CHHABRA, A., AND PRASAD, A. Enabling multi-user controls in smart home devices. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy* (2017), ACM, pp. 49–54.

[91] JIA, Y. J., CHEN, Q. A., WANG, S., RAHMATI, A., FERNANDES, E., MAO, Z. M., PRAKASH, A., AND UNVIERSITY, S. Contexlot: Towards providing contextual integrity to appified iot platforms. In *NDSS* (2017).

[92] JINDOU, J., XIAOFENG, Q., AND CHENG, C. Access control method for web of things based on role and sns. In *Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on* (2012), IEEE, pp. 316–321.

[93] JOHNJFORREST, STEFAN VIGERSKE, H. G. S. E. A. coin-or cbc. https://zenodo.org/record/3700700 (Accessed on 15/5/2020).

[94] KAMMÜLLER, F., NURSE, J. R., AND PROBST, C. W. Attack tree analysis for insider threats on the iot using isabelle. In *International*

Conference on Human Aspects of Information Security, Privacy, and Trust (2016), Springer, pp. 234–246.

[95] KLINEDINST, D. J. Coordinating vulnerabilities in iot devices, January 2016. `https://insights.sei.cmu.edu/cert/2016/01/coordinating-vulnerabilities-in-iot-devices.html` (visited on 28/10/2017).

[96] KREBSONSECURITY. Source code for iot botnet 'mirai' released, Oct. 2016.

[97] KUMAR, H., GHARAKHEILI, H. H., AND SIVARAMAN, V. User control of quality of experience in home networks using sdn. In *Advanced Networks and Telecommuncations Systems (ANTS), 2013 IEEE International Conference on* (2013), IEEE, pp. 1–6.

[98] LAB41. Poseidon: Machine learning, 2017. `https://github.com/Lab41/PoseidonML` (visited on 15/12/2017).

[99] LARA, A., AND RAMAMURTHY, B. Opensec: Policy-based security using software-defined networking. *IEEE Transactions on Network and Service Management 13*, 1 (2016), 30–42.

[100] LEAR. Mud file maker, 10 2017. `https://mudmaker.org/` (visited on 1/11/2017).

[101] LEAR, E., DROMS, R., AND ROMASCANU, D. Manufacturer usage description specification. Tech. rep., IETF Network Working Group, Internet-Draft, 2019.

[102] LEARNING, M. Tom mitchell. *ISBN: 0-07-042807-7, Publisher: McGraw Hill* (1997).

[103] LI, H., WEI, F., AND HU, H. Enabling dynamic network access control with anomaly-based ids and sdn. In *Proceedings of the ACM*

International Workshop on Security in Software Defined Networks & Network Function Virtualization (2019), ACM, pp. 13–16.

[104] LIANG, C.-J. M., KARLSSON, B. F., LANE, N. D., ZHAO, F., ZHANG, J., PAN, Z., LI, Z., AND YU, Y. Sift: building an internet of safe things. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks* (2015), ACM, pp. 298–309.

[105] LOI, F., SIVANATHAN, A., GHARAKHEILI, H. H., RADFORD, A., AND SIVARAMAN, V. Systematically evaluating security and privacy for consumer iot devices. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy* (2017), pp. 1–6.

[106] LUCERO, S. Iot platforms: enabling the internet of things, Mar. 2016.

[107] LUKE, S. *Essentials of Metaheuristics*, second ed. Lulu, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[108] LUO, S., WU, J., LI, J., AND GUO, L. A multi-stage attack mitigation mechanism for software-defined home networks. *IEEE Transactions on Consumer Electronics 62*, 2 (2016), 200–207.

[109] LUPIANA, D., O'DRISCOLL, C., AND MTENZI, F. Defining smart space in the context of ubiquitous computing. *Ubiquitous Computing and Communication Journal 4*, 3 (2009), 516–524.

[110] LYON, G. F. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure. Com LLC (US), 2008.

[111] LYU, M., SHERRATT, D., SIVANATHAN, A., GHARAKHEILI, H. H., RADFORD, A., AND SIVARAMAN, V. Quantifying the reflective ddos attack capability of household iot devices. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (2017), ACM, pp. 46–51.

[112] MADAKAM, S., RAMASWAMY, R., AND TRIPATHI, S. Internet of things (iot): A literature review. *Journal of Computer and Communications 3*, 05 (2015), 164.

[113] MASSIMO, D., ELAHI, M., AND RICCI, F. Learning user preferences by observing user-items interactions in an iot augmented space. In *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization* (2017), pp. 35–40.

[114] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (2008), 69–74.

[115] MCNUTT, R. Supply chain vulnerabilities show weakness in current iot security paradigm. https://www.forbes.com/sites/forbestechcouncil/2020/07/31/supply-chain-vulnerabilities-show-weakness-in-current-iot-security-paradigm (Accessed on 1/8/2020).

[116] MEIDAN, Y., BOHADANA, M., SHABTAI, A., OCHOA, M., TIPPENHAUER, N. O., GUARNIZO, J. D., AND ELOVICI, Y. Detection of unauthorized iot devices using machine learning techniques. *arXiv preprint arXiv:1709.04647* (2017).

[117] MELL, P., SCARFONE, K., AND ROMANOSKY, S. A complete guide to the common vulnerability scoring system version 2.0. In *Published by FIRST-forum of incident response and security teams* (2007), vol. 1, p. 23.

[118] MINERVA, R., BIRU, A., AND ROTONDI, D. Towards a definition of the internet of things (iot). *IEEE Internet Initiative 1*, 1 (2015), 1–86.

[119] MORRISON, J. P. Flow-based programming, 2020. Last accessed 5/7/2020.

[120] MOYANO, R. F., CAMBRONERO, D. F., AND TRIANA, L. B. A user-centric sdn management architecture for nfv-based residential networks. *Computer Standards & Interfaces 54* (2017), 279–292.

[121] MOZILLA. Mozilla webthings gateway. https://iot.mozilla.org/gateway/ (Accessed on 25/4/2020).

[122] MOZILLA. Webthings gateway for raspberry pi user guide. Last accessed 18/7/2020.

[123] MOZILLA. Wot capability schemas. Last accessed 18/8/2020.

[124] MOZILLA IOT. Webthings gateway for raspberry pi, 2020. `https://iot.mozilla.org/docs/gateway-getting-started-guide.html` (accessed on 28/4/2020).

[125] NAEINI, P. E., BHAGAVATULA, S., HABIB, H., DEGELING, M., BAUER, L., CRANOR, L. F., AND SADEH, N. Privacy expectations and preferences in an iot world. In *Thirteenth Symposium on Usable Privacy and Security ({SOUPS} 2017)* (2017), pp. 399–412.

[126] NAGENDRA, V., BHATTACHARYA, A., YEGNESWARAN, V., RAHMATI, A., AND DAS, S. An intent-based automation framework for securing dynamic consumer iot infrastructures. In *Proceedings of The Web Conference 2020* (2020), pp. 1625–1636.

[127] NAH, F. F.-H. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology 23*, 3 (2004), 153–163.

[128] NAYAK, A. K., REIMERS, A., FEAMSTER, N., AND CLARK, R. Resonance: dynamic access control for enterprise networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking* (2009), ACM, pp. 11–18.

[129] NAZERFARD, E., AND COOK, D. J. Using bayesian networks for daily activity prediction. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence* (2013).

[130] NEHRA, A., TRIPATHI, M., AND GAUR, M. Ficur: Employing sdn programmability to secure arp. In *Computing and Communication Workshop and Conference (CCWC), 2017 IEEE 7th Annual* (2017), IEEE, pp. 1–8.

[131] NETO, A. L. M., SOUZA, A. L., CUNHA, I., NOGUEIRA, M., NUNES, I. O., COTTA, L., GENTILLE, N., LOUREIRO, A. A., ARANHA, D. F., PATIL, H. K., ET AL. Aot: Authentication and access control for the entire iot device life-cycle. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM* (2016), ACM, pp. 1–15.

[132] NETWORKS, A. No end in sight for ddos attack size growth, 2017.

[133] NEWMAN, L. H. An elaborate hack shows how much damage iot bugs can do. Last accessed 18 June 2020.

[134] NOBAKHT, M., SIVARAMAN, V., AND BORELI, R. A host-based intrusion detection and mitigation framework for smart home iot using openflow. In *2016 11th International conference on availability, reliability and security (ARES)* (2016), IEEE, pp. 147–156.

[135] NURSE, J. R., CREESE, S., AND DE ROURE, D. Security risk assessment in internet of things systems. *IT professional 19*, 5 (2017), 20–26.

[136] NURSE, J. R., EROLA, A., AGRAFIOTIS, I., GOLDSMITH, M., AND CREESE, S. Smart insiders: exploring the threat from insiders using the internet-of-things. In *2015 International Workshop on Secure Internet of Things (SIoT)* (2015), IEEE, pp. 5–14.

[137] OASIS. Mqtt version 5.0, 2019. Last accessed 5/7/2020.

[138] OF THINGS TOP TEN PROJECT, I. Top 10 iot vulnerabilities (2018), 2018. `https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Top_10` (visited on 28/11/2019).

[139] OF THINGS TOP TEN PROJECT, I. Top 10 iot vulnerabilities (2019), 2019. `https://owasp.org/www-project-internet-of-things/` (visited on 28/1/2020).

[140] ONF. Openflow switch specification version (1.5.1), 2015.

[141] OPENHAB. Raspberry pi, 2020. `https://www.openhab.org/docs/installation/rasppi.html` (accessed on 28/4/2020).

[142] O'LEARY, N., AND CONWAY-JONES, D. Node red, 2013. `https://nodered.org` (visited on 18/10/2018).

[143] PARASURAMAN, R., AND RILEY, V. Humans and automation: Use, misuse, disuse, abuse. *Human factors 39*, 2 (1997), 230–253.

[144] PEDIADITAKIS, D., GOPALAN, A., DULAY, N., SLOMAN, M., AND LODGE, T. Home network management policies: Putting the user in the loop. In *Policies for Distributed Systems and Networks (POLICY), 2012 IEEE International Symposium on* (2012), IEEE, pp. 9–16.

[145] PELTIER, T. R. *Information security risk analysis*. CRC press, 2005.

[146] PERRY, M. Python module for simulated annealing optimization. https://github.com/perrygeo/simanneal (Accessed on 15/1/2020).

[147] PICARD, N., COLIN, J.-N., AND ZAMPUNIERIS, D. Context-aware and attribute-based access control applying proactive computing to iot system. In *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018)* (2018), SCITEPRESS, pp. 333–339.

[148] PIGOZZI, G., TSOUKIAS, A., AND VIAPPIANI, P. Preferences in artificial intelligence. *Annals of Mathematics and Artificial Intelligence 77*, 3-4 (2016), 361–401.

[149] PROJECT, D. Data plane development kit, 2020. Last accessed 15/7/2020.

[150] RASH, M. *Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort*. No Starch Press, 2007.

[151] RASHIDI, P., AND COOK, D. J. Keeping the resident in the loop: Adapting the smart home to the user. *IEEE Transactions on systems, man, and cybernetics-part A: systems and humans 39*, 5 (2009), 949–959.

[152] RAVIDAS, S., LEKIDIS, A., PACI, F., AND ZANNONE, N. Access control in internet-of-things: A survey. *Journal of Network and Computer Applications 144* (2019), 79–101.

[153] RIM, R., AMIN, M. M., AND ADEL, M. Bayesian networks for user modeling: Predicting the user's preferences. In *13th International Conference on Hybrid Intelligent Systems (HIS 2013)* (2013), IEEE, pp. 144–148.

[154] RONEN, E., AND SHAMIR, A. Extended functionality attacks on iot devices: The case of smart lights. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (2016), IEEE, pp. 3–12.

[155] ROSENFELD, A., SINA, S., SARNE, D., AVIDOV, O., AND KRAUS, S. A study of whatsapp usage patterns and prediction models without message content. *arXiv preprint arXiv:1802.03393* (2018).

[156] RUSSELL, S., AND NORVIG, P. Ai a modern approach. *Learning 2*, 3 (2005), 4.

[157] RUSSELL, S. J., AND NORVIG, P. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[158] SCHNEIDER, F. B. Least privilege and more [computer security]. *IEEE Security & Privacy 1*, 5 (2003), 55–59.

[159] SCHNEIER, B. Your wifi-connected thermostat can take down the whole internet. we need new regulations, 2016. `https://www.washingtonpost.com/posteverything/wp/2016/11/03/` (visited on 8/1/2018).

[160] SCHUSTER, R., SHMATIKOV, V., AND TROMER, E. Situational access control in the internet of things. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), pp. 1056–1073.

[161] SERROR, M., HENZE, M., HACK, S., SCHUBA, M., AND WEHRLE, K. Towards in-network security for smart homes. In *Proceedings of the 13th International Conference on Availability, Reliability and Security* (2018), pp. 1–8.

[162] SETHI, P., AND SARANGI, S. R. Internet of things: architectures, protocols, and applications. *Journal of Electrical and Computer Engineering 2017* (2017).

[163] SHERWOOD, R., CHAN, M., COVINGTON, A., GIBB, G., FLAJSLIK, M., HANDIGOL, N., HUANG, T.-Y., KAZEMIAN, P., KOBAYASHI, M., NAOUS, J., ET AL. Carving research slices out of your production networks with openflow. *ACM SIGCOMM Computer Communication Review 40*, 1 (2010), 129–130.

[164] SHIN, S., XU, L., HONG, S., AND GU, G. Enhancing network security through software defined networking (sdn). In *Computer Communication and Networks (ICCCN), 2016 25th International Conference on* (2016), IEEE, pp. 1–9.

[165] SIKDER, A. K., BABUN, L., CELIK, Z. B., ACAR, A., AKSU, H., MCDANIEL, P., KIRDA, E., AND ULUAGAC, A. S. Multi-user multi-

device-aware access control system for smart home. *arXiv preprint arXiv:1911.10186* (2019).

[166] SIVANATHAN, A., SHERRATT, D., GHARAKHEILI, H. H., RADFORD, A., WIJENAYAKE, C., VISHWANATH, A., AND SIVARAMAN, V. Characterizing and classifying iot traffic in smart cities and campuses. *dvanced Networks and Telecommunications Systems (ANTS)* (2017).

[167] SIVANATHAN, A., SHERRATT, D., GHARAKHEILI, H. H., SIVARAMAN, V., AND VISHWANATH, A. Low-cost flow-based security solutions for smart-home iot devices. In *Advanced Networks and Telecommunications Systems (ANTS), 2016 IEEE International Conference on* (2016), IEEE, pp. 1–6.

[168] SIVARAMAN, V., CHAN, D., EARL, D., AND BORELI, R. Smartphones attacking smart-homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks* (2016), ACM, pp. 195–200.

[169] SIVARAMAN, V., GHARAKHEILI, H. H., VISHWANATH, A., BORELI, R., AND MEHANI, O. Network-level security and privacy control for smart-home iot devices. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on* (2015), IEEE, pp. 163–167.

[170] SMARTTHING. Local processing. https://support.smartthings.com/hc/en-us/articles/209979766-Local-processing (Accessed on 15/4/2020).

[171] SMARTTHINGS. How can i invite members in the smartthings app? https://support.smartthings.com/hc/en-us/articles/115002085066-How-can-I-invite-members-in-the-SmartThings-app- (Accessed on 15/1/2020).

[172] SØRENSEN, D. A., VANGGAARD, N., AND PEDERSEN, J. M. Automatic profile-based firewall for iot devices. Master's thesis, Aalborg University, Denmark, 2017.

[173] SPRING, T. Iot insecurity: Pinpointing the problems, July 2016. `https://threatpost.com/iot-insecurity-pinpointing-the-problems/119389/2/` (visited on 2/12/2017).

[174] STANDARD, O. extensible access control markup language (xacml) version 3.0, 2005.

[175] STANISLAV, M., AND BEARDSLEY, T. Hacking iot a case study on baby monitor exposures and vulnerabilities. Tech. rep., RAPID7, 2015.

[176] STUCKEY, P. J. Lazy clause generation: Combining the power of sat and cp (and mip?) solving. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming* (2010), Springer, pp. 5–9.

[177] SUNDARESAN, S., BURNETT, S., FEAMSTER, N., AND DE DONATO, W. Bismark: A testbed for deploying measurements and applications in broadband access networks. In *USENIX Annual Technical Conference* (2014), pp. 383–394.

[178] SZYDLO, T., BRZOZA-WOCH, R., SENDOREK, J., WINDAK, M., AND GNIADY, C. Flow-based programming for iot leveraging fog computing. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2017 IEEE 26th International Conference on* (2017), IEEE, pp. 74–79.

[179] TEAM, S. Stringify, 2014. `https://www.stringify.com/` (visited on 8/10/2018).

[180] TIAN, L., FAMAEY, J., AND LATRÉ, S. Evaluation of the ieee 802.11 ah restricted access window mechanism for dense iot networks. In *2016 IEEE 17th international symposium on a world of wireless, mobile and multimedia networks (WoWMoM)* (2016), IEEE, pp. 1–9.

[181] TIAN, Y., ZHANG, N., LIN, Y.-H., WANG, X., UR, B., GUO, X., AND TAGUE, P. Smartauth: User-centered authorization for the internet of things. In *26th {USENIX} Security Symposium ({USENIX} Security 17)* (2017), USENIX Association, pp. 361–378.

[182] TP-LINK. How to set up firewall to restrict internet activity on td-w9970 (new logo)? Last accessed 18 June 2020.

[183] TP LINK. How to set access control of the internet with firewall on modem router (self-developed ui)?, May 2016. `http://www.tp-link.com/us/faq-467.html` (visited on 6/10/2017).

[184] TRABELSI, Z. Microsoft windows vs. apple mac os x: Resilience against arp cache poisoning attack in a local area network. *Information Security Journal: A Global Perspective 25*, 1-3 (2016), 68–82.

[185] TRIMANANDA, R., YOUNIS, A., WANG, B., XU, B., DEMSKY, B., AND XU, G. Vigilia: Securing smart home edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)* (2018), IEEE, pp. 74–89.

[186] VORONKOV, A., IWAYA, L. H., MARTUCCI, L. A., AND LINDSKOG, S. Systematic literature review on usability of firewall configuration. *ACM Computing Surveys (CSUR) 50*, 6 (2017), 1–35.

[187] WANG, Q., DATTA, P., YANG, W., LIU, S., BATES, A., AND GUNTER, C. A. Charting the attack surface of trigger-action iot platforms. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019), pp. 1439–1453.

[188] WARD, R., AND BEYER, B. Beyondcorp: A new approach to enterprise security. *login 39* (2014), 5–11.

[189] WILSON, H. J., SHAH, B., AND WHIPPLE, B. How people are actually using the internet of things. *Harvard Business Review* (2015), 1–6.

[190] WOOD, D., APTHORPE, N., AND FEAMSTER, N. Cleartext data transmissions in consumer iot medical devices. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy* (2017), pp. 7–12.

[191] WU, Z.-H., LIU, A., ZHOU, P.-C., AND SU, Y. F. A bayesian network based method for activity prediction in a smart home system. In *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on* (2016), IEEE, pp. 001496–001501.

[192] XU, K., WANG, F., AND JIA, X. Secure the internet, one home at a time. *Security and Communication Networks 9*, 16 (2016), 3821–3832.

[193] YAHYAZADEH, M., PODDER, P., HOQUE, E., AND CHOWDHURY, O. Expat: Expectation-based policy analysis and enforcement for appified smart-home platforms. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies* (2019), pp. 61–72.

[194] YANG, H., LEE, W., AND LEE, H. Iot smart home adoption: the importance of proper level automation. *Journal of Sensors 2018* (2018).

[195] YIAKOUMIS, Y., KATTI, S., HUANG, T.-Y., MCKEOWN, N., YAP, K.-K., AND JOHARI, R. Putting home users in charge of their network. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (2012), ACM, pp. 1114–1119.

[196] YU, T., SEKAR, V., SESHAN, S., AGARWAL, Y., AND XU, C. Handling a trillion (unfixable) flaws on a billion devices: Rethinking net-

work security for the internet-of-things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks* (2015), ACM, p. 5.

[197] ZENG, E., MARE, S., AND ROESNER, F. End user security and privacy concerns with smart homes. In *Thirteenth Symposium on Usable Privacy and Security ({SOUPS} 2017)* (2017), pp. 65–80.

[198] ZENG, E., AND ROESNER, F. Understanding and improving security and privacy in multi-user smart homes: A design exploration and in-home user study. In *28th {USENIX} Security Symposium ({USENIX} Security 19)* (2019), pp. 159–176.

[199] ZHENG, S., APTHORPE, N., CHETTY, M., AND FEAMSTER, N. User perceptions of smart home iot privacy. *Proceedings of the ACM on Human-Computer Interaction 2*, CSCW (2018), 1–20.

[200] ZHOU, W., JIA, Y., YAO, Y., ZHU, L., GUAN, L., MAO, Y., LIU, P., AND ZHANG, Y. Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms. In *28th {USENIX} Security Symposium ({USENIX} Security 19)* (2019), pp. 1133–1150.

[201] ZURKO, M. E. User-centered security: Stepping up to the grand challenge. In *21st Annual Computer Security Applications Conference (ACSAC'05)* (2005), IEEE, pp. 14–pp.