Master’s Programme in Security and Cloud Computing

# Automatic Ownership Change Detection for IoT devices

Artur Valiev

Aalto University

MASTER’S THESIS

Aalto University
**MASTER'S THESIS** 2018

# Automatic Ownership Change Detection for IoT devices

**Artur Valiev**

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.
Otaniemi, July 2, 2018

| | |
|---|---|
| Supervisor: | Professor N. Asokan, Aalto University |
| | Associate Professor Livshitz I.I., ITMO University |
| Advisor: | Dr. Samuel Marchal (Postdoc), Aalto University |

**Aalto University**
**School of Science**
**Master's Programme in Security and Cloud Computing**

Aalto University

**Abstract**

**Aalto University, P.O. Box 11000, FI-00076 Aalto** www.aalto.fi

**Author**
Artur Valiev

**Title**
Automatic Ownership Change Detection for IoT devices

**School** School of Science

**Master's programme** Computer, Communication and Information Sciences

**Major** Security and Cloud Computing **Code** SCI3084

**Supervisor** Professor N. Asokan, Aalto University, Professor Livshitz I.I., ITMO University

**Advisor** Dr. Samuel Marchal (Postdoc), Aalto University

**Level** Master's thesis **Date** July 2, 2018 **Pages** 64 **Language** English

## Abstract

Considering the constant increases in Internet Of Things (IoT) smart home devices prevalence, their ownership is likely to change. This introduces novel privacy issues. Smart home devices store owner's sensitive information, which needs to be handled securely in case of change in device ownership. Currently employed smart home devices cannot detect changes in their ownership, which raises a great number of privacy and security issues. To address this problem, we propose a system called *FoundIoT* for automatic detection of IoT device ownership change. *FoundIoT* provides a technique to detect change of ownership based on device context, which is inferred by monitoring wireless communication channels. Finally, we present a prototype implementation of *FoundIoT* for the proposed automatic ownership change detection technique. We show that *FoundIoT* achieves a satisfactory performance. The implementation is supported by a wide range of IoT devices and demonstrates a high speed (up to 1 minute 39 seconds) and 100% accuracy of ownership change detection.

**Keywords** Internet Of Things, Ownership Change, Smart Home

# Acronyms

| | |
|---|---|
| AP | Access Point |
| DHCP | Dynamic Host Configuration Protocol |
| HIDs | Human Interface Devices |
| IoT | Internet Of Things |
| JSON | JavaScript Object Notation |
| LAN | Local Area Network |
| NFC | Near field communication |
| NIC | Network Interface Card |
| RFID | Radio Frequency Identification |
| STA | Wireless Station |
| WLAN | Wireless Local Area Network |

# Contents

**Abstract** ii

**List of Acronyms** iii

**Contents** iv

**1. Introduction** 1

1.1 Motivation . . . . . . . . 1

1.2 Contributions . . . . . . . . 1

1.3 Organization . . . . . . . . 2

**2. Background** 3

2.1 Internet Of Things (IoT) . . . . . . . . 3

2.2 Communication technologies . . . . . . . . 4

2.2.1 WiFi . . . . . . . . 4

2.2.2 Bluetooth . . . . . . . . 4

2.3 Network Traffic Monitoring . . . . . . . . 5

2.3.1 Network Card Operation Modes . . . . . . . . 5

2.3.2 Monitoring Tools . . . . . . . . 6

2.4 Context Awareness . . . . . . . . 7

2.5 chownIoT system . . . . . . . . 8

2.6 Technical Background . . . . . . . . 9

2.6.1 Raspberry Pi . . . . . . . . 9

2.6.2 Bash scripting . . . . . . . . 9

2.6.3 Python . . . . . . . . 10

**3. Problem Statement** 11

3.1 Description . . . . . . . . 11

3.2 Adversary Model . . . . . . . . 12

3.3 Requirements . . . . . . . . 13

**4. System Design** **15**
4.1 Solution Overview . . . . . 15
4.2 Design Choices . . . . . 17
4.2.1 Context Identification . . . . . 17
4.2.2 Data recording . . . . . 18
4.2.3 Multistage Ownership Change Detection . . . . . 18
4.2.4 Statistical analysis . . . . . 18
4.3 Components . . . . . 19
4.3.1 WiFi context monitor . . . . . 19
4.3.2 Bluetooth context monitor . . . . . 20
4.3.3 Time Series Generator . . . . . 20

**5. Ownership Change Detection** **22**
5.1 Time Series Analysis . . . . . 22
5.1.1 Jaccard Index . . . . . 23
5.1.2 Kullback-Leibler divergence . . . . . 24
5.2 Captured Data Filtering . . . . . 26
5.3 Analysis Conclusions . . . . . 26

**6. Implementation** **29**
6.1 Smart Home Device . . . . . 29
6.2 WiFi Context Monitoring . . . . . 30
6.3 Bluetooth Context Monitoring . . . . . 30
6.4 Time Series Generation . . . . . 30
6.5 Time Series Analysis . . . . . 31

**7. Evaluation** **32**
7.1 Deployability . . . . . 32
7.2 Resource Constraints . . . . . 32
7.3 Performance . . . . . 35
7.3.1 Ownership Change Scenarios Description . . . . . 35
7.3.2 Accuracy Evaluation . . . . . 41
7.3.3 Speed Evaluation . . . . . 44
7.4 Re-evaluation using AP Detection technique . . . . . 45
7.4.1 Experiment setup . . . . . 45
7.4.2 Accuracy Evaluation . . . . . 46
7.4.3 Speed evaluation . . . . . 48
7.5 Security . . . . . 48
7.5.1 MAC Address Spoofing . . . . . 48

7.5.2 Ownership Change Detection Delay . . . . . . . . . . . 49
7.5.3 Susceptibility to Power Cuts . . . . . . . . . . . . . . . . 49

**8. Related work 51**
8.1 Smart Home Device Privacy . . . . . . . . . . . . . . . . . . . . . . . 51
8.2 Context Aware Security . . . . . . . . . . . . . . . . . . . . . . . . . 52

**9. Conclusions 55**
9.1 Summary of Contributions . . . . . . . . . . . . . . . . . . . . . . . 55
9.2 Future work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

**Bibliography 57**

**List of Figures 62**

**List of Tables 64**

# 1. Introduction

## 1.1 Motivation

The IoT is considered as the next step in the evolution of connected devices after personal computers and portable devices. These systems include Smart Home devices, wearables, connected cars, industrial IoT and others.

With different IoT ecosystems widespread, critical security and privacy concerns arise and need to be addressed. Certain IoT devices have an ability to collect private and confidential information about their owners, which raises a variety of privacy and security issues. Such information include sensitive data, such as user profile data, authentication details, network configuration settings and credentials.

The current extensive prevalence of Smart Home devices has raised an issue of ownership change. Such situations might happen in case of selling, borrowing or even theft of IoT devices.

Ownership change can lead to leakage of the previous owner's sensitive information stored on the IoT device, as a new owner gains full access to it. Additionally, the previous owner of the IoT device could have remote access to it, which can lead to leakage of the new owner's data stored on the device or its cloud storage.

Therefore, a technique for an IoT device to detect on its own if its owner has changed is required to handle such situations in a secure manner.

## 1.2 Contributions

The major contributions of this work are:

- **An automatic ownership change detection system called *FoundIoT*:**
  - Developing an automatic technique that allows the IoT device to scan its vicinity for surrounding connected devices and detect changes in context

using wireless communication channels (presented in Chapter 4).

- **Data mining techniques:**
  - Developing data mining techniques applied to the collected wireless communication data (presented in Chapter 5).

- **A prototype implementation of the system:**
  - Developing a prototype implementation of the proposed system on the Raspberry Pi 3 (presented in Chapter 6). We use Python programming language for data formatting and analysis, and Bash scripting for scheduling and data collection.

- **Evaluation:**
  - Providing the system evaluation in terms of its performance and security (presented in Chapter 7). The implemented prototype is able to detect ownership change with 100% accuracy and speed up to 1 minute 39 seconds. The prototype can be realized on a wide variety of off-the-shelf IoT devices and provides partial resilience to adversaries. The complete resilience can be achieved with existing techniques described in Section 7.5.

## 1.3 Organization

The remainder of this manuscript is organized as follows: **Chapter 2** provides an overview of IoT and smart home devices, and technical background to introduce the software and technologies used in our solution. **Chapter 3** presents the adversary model for IoT device ownership change. We also discuss the requirements for the proposed system to protect against the specified adversary. **Chapter 4** provides an overview of the proposed solution and also discusses its design and components. **Chapter 5** describes the details of ownership change detection. **Chapter 6** provides the implementation details of the proposed solution. **Chapter 7** presents the evaluation of the system, by measuring its accuracy and speed of detection. **Chapter 8** discusses the related work. Finally, in **Chapter 9**, we conclude the work and share thoughts on future development of the system.

# 2. Background

## 2.1 Internet Of Things (IoT)

IoT represents a general concept for the ability of network devices to sense and collect data from the world around them, and interact with the physical world. IoT devices share data across the Internet where it can be processed and utilized for various purposes. IoT provides connectivity across devices used in everyday routine, such as smartphones, washing machine, television set, microwave and coffee maker. Generally, they are connected to the Internet, however some IoT devices can also act as a gateway for other devices and connect directly to them. IoT is a giant network connecting almost everything.

The term "Internet of Things" was firstly mentioned by Kevin Ashton, co-founder and executive director of the Auto-ID Center at MIT, in 1999. He anticipated a major potential for IoT technology, including the ability of computer systems to automatically collect and analyze data with much higher accuracy compared to manual human operations. This leads to cost and human error reduction and provides up-to-date information on the state of analyzed environment. Currently, IoT is involved in a vast variety of industries, such as transportation, agriculture, health-care, security solutions and entertainment. One of the IoT industries with rapidly increasing prevalence is "smart home".

Smart home is a convenient home setup where appliances and devices, such as door locks, thermostats, home monitors, cameras and lights, can be automatically controlled remotely from anywhere in the world through the Internet. All these devices generally operate in home environments. Therefore, they deal with sensitive personal data about their owners on a daily basis. This raises a vast variety of security and privacy concerns and makes smart home devices the target for attacks from an adversary [1].

## 2.2 Communication technologies

IoT devices can be enabled with a vast array of technologies available to vendors. Devices can use Near field communication (NFC) and Radio Frequency Identification (RFID) for low range communication (up to 50 cm); and WiFi, ZigBee, Bluetooth, Bluetooth LE [2], Z-Wave [3], IoTivity [4] and Homekit [5] for medium range (up 100 m). Each mentioned communication standard provides sufficient connectivity for a device independently, which means that an IoT device can use one of them or several of them.

The following sections 2.2.1 and 2.2.2 describe two prevalent communication technologies used in IoT devices currently.

### 2.2.1 WiFi

WiFi is a Wireless Local Area Network (WLAN) technology that uses radio waves to transmit data. It is based on IEEE 802.11 standards [6]. The IEEE 802.11 standard describes two wireless network modes: *infrastructure* and *ad hoc*. In case of ad hoc operation mode, devices establish peer-to-peer communication and exchange data directly, without any intermediate node. On the contrary, infrastructure mode requires a special node in the network called Access Point (AP) that all devices in the network connect to. It also serves as a communication bridge between devices in local network and other nodes in the Internet.

In general, IoT devices enable infrastructure mode during initial configuration and await connection request from *a control device*, which could be a smartphone, tablet or laptop. The IoT device creates a personal wireless AP, and the control device gets connected to it for establishing the communication and credentials exchange. The control device provides information on itself and the base wireless AP. Once the connection is configured, the IoT device disables its own AP and switches to *managed mode* to connect to the base AP.

### 2.2.2 Bluetooth

Bluetooth is a short-range wireless communication technology that uses radio signals to send and receive data. It was originally designed as a replacement for cable connection for various Human Interface Devices (HIDs) and soon became a widely adopted wireless connectivity solution. Bluetooth operates in the 2.4 GHz industrial, scientific, and medical band. The technology is maintained by the Bluetooth Special Interest Group (SIG).

IoT devices use Bluetooth in different scenarios. First, it can be used as the primary communication channel. Second, Bluetooth is used for initial configuration routines. In this case, it serves as a secondary communication channel and is used for initial connection establishment. The control device acts as a master, and the IoT device joins the network as a slave. Once the initial Bluetooth pairing process is successfully completed, devices can exchange the necessary information for future communication, such as WiFi network and cloud service credentials. If the IoT device plays the role of a gateway, the control device can instruct it to form another network for additional devices. In this network, the IoT device acts as a master and other modules are connected to the network as slaves.

A major benefit of Bluetooth pairing procedures is that they are performed only once, during the initial connection. After that, devices from one network trust each other and can connect automatically for later communications.

## 2.3 Network Traffic Monitoring

Network traffic monitoring is the process of analyzing traffic patterns used for assessing network performance, resource availability and security. Network monitoring incorporates packet sniffing and capturing techniques and generally requires inspecting each incoming and outgoing packet.

In order to analyze traffic, the Network Interface Card (NIC) should support specific operation modes. A NIC is an electronic component that connects a computer to a network. The NIC provides communication using specific physical and data link layer standards, such as Ethernet and WiFi. This provides a base for full network protocol stack, allowing communication among small groups of computers on the same Local Area Network (LAN) and large-scale network communication through routable protocols, such as Internet Protocol (IP). The NIC allows computers to communicate over a computer network, either by using a wired or wireless connection.

### 2.3.1 Network Card Operation Modes

In general, manufacturers provide the support for several operation modes to the NIC. The card mode depends on the current tasks assigned to it.

Most network cards support the following operation modes:

**Managed**

Managed mode is sometimes referred to as client mode. It is the most commonly used and supported operation mode among network cards. The interface in managed mode requires a master to operate. It joins the network, and the device gets associated with it. Nodes in managed mode do not communicate with each other directly, but only via the associated master.

**Master**

*Master mode* enables a wireless station to operate as an AP. It allows other nodes to connect to the station and join its network.

**Ad-hoc**

Another commonly supported mode is ad-hoc. In this mode, wireless stations create a direct peer-to-peer communication. Ad-hoc mode is used for temporary connection establishing and data exchange in cases of AP unavailability or inoperability.

**Monitor & Promiscuous**

Finally, wireless cards support *promiscuous* and *monitor mode* functionality. In these modes, the network card stops transmitting the data and starts listening on a specific channel or devices connected to a local network. The major difference is that promiscuous mode operates only when connected to a wired or wireless LAN. It allows the card to sniff frames sent by other members of a local network. On the other hand, the monitor mode does not require a connection to an AP. The network card sniffs frames sent in any nearby network on a specified frequency. In networks, such as Ethernet and IEEE 802.11, each frame includes source and destination MAC addresses. In managed mode, the NIC only accepts incoming frames that are addressed to its MAC address and broadcast/multicast frames. Other frames are dropped. In promiscuous/monitor mode, however, the NIC accepts all frames, thus allowing the node to read frames intended for other machines and network devices. Thus, promiscuous and monitor modes are very useful in network monitoring and troubleshooting tasks. They are used to locate malfunctions and abnormal activity in the network.

### 2.3.2 Monitoring Tools

As it was mentioned earlier, traffic monitoring is performed for network analysis and security assessment. In order to analyze network traffic, specialized tools are required.

One of the most popular Internet frames capturing tools is Tcpdump [7]. Tcpdump is an open source command-line tool for monitoring (sniffing) network traffic. It is one of the most powerful tools for analyzing network traffic. Tcpdump works by capturing and displaying packet headers and matching them against a set of filters. It understands boolean search operators and can use host names, IP addresses, network names, and protocols as arguments.

Tcpdump operates in two modes. First, it can analyze traffic flow in real time, displaying all packets that match a specified filter. This mode is useful for live traffic analysis when it is required to constantly monitor traffic and register expected behavior as soon as it happens.

Another mode is used for offline traffic analysis. Tcpdump can capture packets and save them to a .pcap file. This mode allows to analyze a captured stream of frames in detail, apply different filters and extract the required data. This data can be used for many purposes such as security & forensic analyses, troubleshooting & network administration or to understand and learn the network.

As an alternative, it may be more suitable to look at graphical tools, such as Wireshark [8]. Such tools often provide a more workable tool-set for looking at larger volumes of traffic. Wireshark also provides human readable filter building tools that can be a time saver. However, it is computationally less efficient than Tcpdump and is not usually used for automated tasks.

## 2.4 Context Awareness

Context awareness can be defined as the ability for a system or its components to gather information about its environment at any given time and adapt its behavior accordingly. Context includes any information that is relevant to a given entity, such as a person, a device or an application. As such, a wide range of categories including time, location, device, identity, activity and nearby devices/users fall into the term of contextual information.

With the rapid development of IoT, the amount of devices that provide sensing capabilities has increased. Most smart home devices are equipped with various sensors that allow them to measure temperature, humidity and pressure; identify location and identity of the user who requests access. A smart home device with temperature sensor can detect variations of room temperature. Whenever a change is detected, the device sends a request for adjusting temperature to the home automation system.

Another example is related to data security. The ConXsense framework [9] provides a context-aware control system for mobile devices. It classifies contexts automatically with the use of context sensing and machine learning techniques and adjusts the security and privacy-related properties.

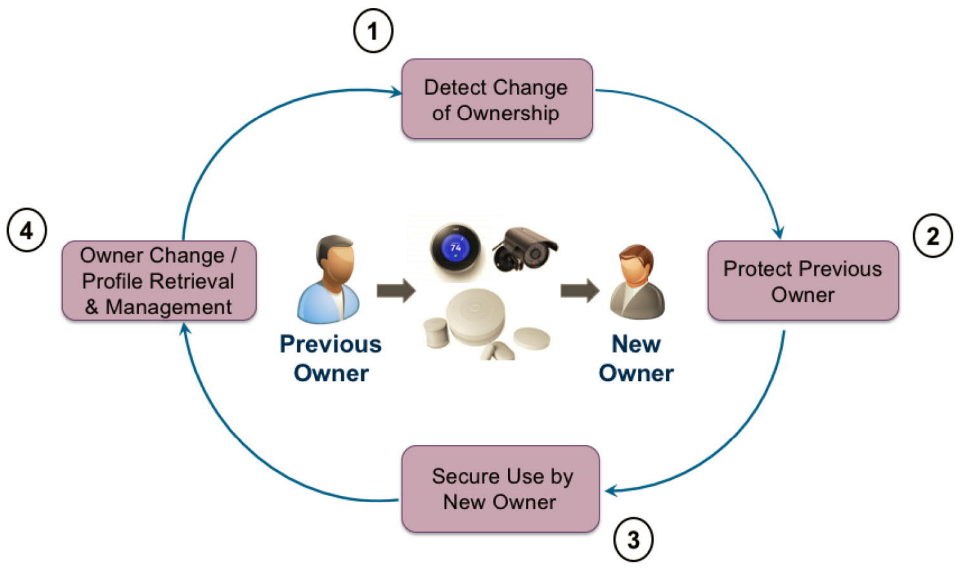

**Figure 2.1.** chownIoT Flow Diagram [10]

## 2.5 chownIoT system

Contextual information can be used to improve the system security. IoT devices used in households contain owner's sensitive information, which can be leaked or stolen during ownership change.

The chownIoT system developed by Khan [10] presents a technique for secure handling of smart home device ownership change. The author proposes a privacy protocol to tackle the issues of authentication and provides data protection techniques, such as data encryption, to ensure the owner's privacy. The protocol ensures data privacy during ownership change by preserving owner's contexts on the device with encryption. It does not rely on any specific hardware dependencies. Therefore, the system can be implemented on a variety of off-the-shelf devices.

chownIoT also describes an owner's profile management scheme to effectively manages ownership during the IoT device life cycle. To access or retrieve the profile, the user needs to present proper credentials. The scheme also provides a simple and reliable technique to transfer device ownership. This could be useful in case of device lending for a limited time.

Figure 2.1 describes the main steps of chownIoT. Once the system is set up, it launches the ownership change detection procedure. If the ownership change is detected, chownIoT tests the current context by searching for a trusted device in the vicinity and sending a challenge to it.

If the system receives a valid response to the challenge, it creates a new context for the current owner. Otherwise, chownIoT locks the profile and starts looking for a new control device. Once it is found, the smart home device provides a list of available profiles to it.

When a new control device accesses the IoT device, chownIoT provides it with a list of encrypted user profiles and an option to create a new profile. A new user can choose an existing profile or create a new one. If the user attempts to access an existing profile, the IoT device verifies it with the credential mechanism set up by profile owner. The system support for trusted and new control devices allows a convenient access to the IoT device and improves the usability. At the same time, the differentiation between trusted and new control devices provides a fine-grained access to the smart home device.

## 2.6 Technical Background

This section provides an overview of software and hardware solutions used in implementation of the project.

### 2.6.1 Raspberry Pi

A Raspberry Pi [11] is a credit card-sized computer created by Eben Upton, the founder of the Raspberry PI Foundation, and originally designed for education. The creator's goal was to help students improve programming skills and hardware awareness. However, it became extremely popular among beginner developers, enthusiasts and researchers thanks to its low price and small size.

The Raspberry Pi was originally designed for the Linux-based operating systems, and many Linux distributions now have a version optimized for it. Two of the most popular options are Raspbian and Pidora, which are based on the Debian and Fedora operating systems, respectively.

The Raspberry Pi is a resource-constrained device, however it provides all features of a common Linux machine. Several generations of Raspberry PI were released to the market. The latest available one is the 3rd generation of Raspberry PI. The major difference of current generation is the support of on board Bluetooth and WiFi adapters, which are widely used in most IoT devices.

### 2.6.2 Bash scripting

Bash [12] is a command line interface for interacting with the operating system. It is widely used on a vast variety of GNU/Linux distributions and Unix systems, such as Mac OS X. Bash shell was created by the Free Software Foundation in the late 1980s and was intended as a free software alternative to the Bourne shell. It provided all the features of this shell and introduced new features, such as integer arithmetic and job control.

Bash provides several modes of execution to the user. In the interactive mode, the shell gives immediate feedback for user input. In addition, Bash provides command scripting mode, also known as Bash shell scripting. In this mode, the user can create complex programs that consist of various commands and constructs, including loops, conditions and functions.

Bash scripts are widely used in administrative and scheduling tasks as well. These tasks can be launched automatically, depending on various conditions, such as system boot, specific user input, system or network event.

### 2.6.3 Python

Python [13] is an object-oriented, high-level programming language. It provides clean and readable syntax that makes it easy to learn.

Additionally, Python supports modularity and packaging. This allows to create Python modules that can be used as external packages across various projects. Python provides straightforward techniques for adding and integrating new modules to the existing applications.

Another benefit of Python is the free of charge availability of the interpreter and standard libraries. Python and all necessary dependencies are available on all major platforms. Thus, it is widely used in various projects by developers, manufacturers and researchers for low-cost prototype development and research projects.

# 3. Problem Statement

In this chapter, we specify the problem that we are aiming to solve. It relates to various potential threats during the ownership change process. Section 3.1 gives the description of the problem, Section 3.2 outlines the adversary model and finally, Section 3.3 specifies system requirements for mitigating identified threats.

## 3.1 Description

In Chapter 1, we mentioned the aspect of personal data handling by IoT devices. Typically, smart home devices store various sensitive information, including device readings, owner's account preferences and network credentials. In most cases, this sensitive information is stored in the device memory and on its cloud platform with no proper security mechanisms [14]. This introduces various threats against sensitive information related to the owner.

IoT device ownership might change if the current owner decides to sell or lend it. Such situations raise a variety of security questions, such as the device ability to detect ownership change and protect the previous owner's data stored in its memory and the cloud service. Device also needs to establish proper access control rules for both parties: the previous owner and the new one.

Inadequate security measures for mitigating potential threats provide a possibility for an attacker to gain access to the owner's sensitive information. Thus, an automatic ownership change detection algorithm is required for IoT devices. Its primary objective is to detect the ownership change and take appropriate measures for mitigating possible attacks on the previous owner's personal data. Potential attackers in ownership change situations are the previous owner and a new owner of the device located in various environments. Hence, our solution needs to differentiate and identify different ownership change scenarios to apply proper threat mitigation in each case.

Ownership change could happen in three major scenario categories: selling a device, lending a device for a limited amount of time and device stealing. We specify potential threats for the IoT device ownership change process by performing the adversary modeling.

A system for secure handling of the contextual data stored on the smart home device has already been described by Khan in his MSc Thesis [10]. His work focused on protecting the device owner's privacy and specifying owner's profile management scheme to improve security of the device during its life cycle. Meanwhile, our work is focused on another part in the device ownership management, the detection of change in ownership. While Khan presented a simple detection technique in his thesis, we propose an advanced automatic mechanism for ownership change detection to provide a proper level of security to the system.

## 3.2 Adversary Model

This section specifies the adversary model for smart home IoT device ownership change by presenting attacker goals, attack surface and capabilities of a potential attacker who is attempting to perform malicious activities targeted on the system.

**Attacker goals**: The primary objective of an attacker is to gather sensitive data stored on the IoT device or its cloud storage. This data includes AP credentials that are saved in the smart home device storage, and collected sensoric data that is sent to the control device or a cloud storage.

**Attack surface**: Due to lack of proper security mechanisms, IoT devices provide an extensive attack surface. We focus on the most critical part of it, which composes the primary set of vulnerabilities.

**Device Memory**

Generally, IoT devices store sensitive owner's data (usernames and passwords, third-party and network credentials, encryption keys, and information collected by the device) in their local storage in an insecure manner and provide no tools for data integrity checking. Furthermore, smart home devices are shipped with hardcoded credentials for ease of setup [15].

**Cloud storage**

Some smart devices provide cloud service functionality. Owners can access data and control their smart home devices remotely. Cloud services that provide basic security techniques leave the possibility for a great number of remote web attacks open for adversaries.

**Device Web Interface**

Some IoT devices provide a web interface for accessing its features and local device management. Some web interfaces lack proper security mechanisms, leading to the possibility of various vulnerabilities, including SQL injections, cross-site scripting, username enumeration and account lockout. Costin et al. [14] performed the analysis of popular IoT firmware distributions and reported that 24% of the web interfaces provide serious vulnerabilities.

**Attacker capabilities**: Our threat model considers *a new owner* and *the previous owner* of the device as potential attackers. During ownership change, a new owner receives full control on the acquired device. A new owner can attempt to steal the data stored in the device memory, by simulating the previous owner's environment. This can be achieved by spoofing the wireless network settings and mislead the system security mechanisms, which will result in the previous owner's sensitive data leakage. Due to direct physical access to the device, a new owner is able to directly extract data stored in the local memory. Apart from these, a new owner can intercept the device network activity and control its communication. In contrast, the previous owner can access the data stored in the IoT device cloud storage remotely, as he knows the cloud credentials of the device. In addition, some IoT devices provide the functionality of remote management and control, which can be exploited by the previous owner of the device.

## 3.3 Requirements

The prime objective of this project is to design a technique for automatic IoT device ownership change detection. The proposed solution should automatically detect ownership change and provide an adequate accuracy.

In this section, we set requirements for the proposed solution.

**R1 - Deployability**

The goal of the project is to develop a universal automatic ownership change detection system for IoT devices. Thus, the proposed solution needs to be supported by a wide range of IoT devices.

**R2 - Efficient execution under resource constraints**

Most IoT devices provide limited computing resources. The system should operate on devices with limited resources and should not lead to high system load. The set of minimum hardware specifications is listed below. They are based on Tessel 2 development board [16].

- RAM: from 64 MB

- CPU: from 500 MHz
- Persistent storage: from 32 MB

**R3 - Performance**

**R3.1 - Accuracy**

The system needs to detect ownership change with at least 95% accuracy and false positive rate lower than 1%.

**R3.2 - Speed**

The solution needs to detect ownership change during the IoT device initial setup process, which typically takes up to 2 minutes [17].

The specified performance criteria inversely correlate with each other. Achieving more accurate results will lead to decrease in speed. Thus, the proposed solution should balance these criteria in order to provide adequate performance.

**R4 - Security**

The proposed solution needs to be resilient to the adversary model described in Section 3.2. The solution should prevent attacker attempts of detection disruption, by mitigating spoofing of network settings and collected data stored on the device.

# 4. System Design

## 4.1 Solution Overview

In Chapter 3, we identified security threats that might occur during ownership change of an IoT device. In addition, we determined the requirements to achieve proper system accuracy. Now, we describe a new system called *FoundIoT* (Ownership Change Detection system) aimed to mitigate the security issues of ownership change and reach the specified requirements. *FoundIoT* provides a reliable technique that detects device ownership change in an automatic manner. The complete system consists of two components: *FoundIoT* and chownIoT [10]. *FoundIoT* provides the functionality of automatic ownership change detection for the IoT device. chownIoT manages ownership change by ensuring data integrity and providing user privacy enhancements.

*FoundIoT* is integrated in the IoT device itself. It allows the device to detect change of its owner on its own. The device does not require any additional, external tools to detect ownership change.

*FoundIoT* assumes changes in context to detect ownership change of an IoT device. We define *context* as a set of active wireless nodes that send or transmit data in the device vicinity. This wireless activity characterizes the IoT device context.

The flow diagram of *FoundIoT* is depicted in Figure 4.1. During the first stage, the system collects data about the IoT device context, it identifies every network connected device in its vicinity. *FoundIoT* collects the contextual data periodically and saves captured data in a time series format. Each observation consists of timestamp and the collected context at the corresponding time.

Once the data is collected, the system proceeds to the next stage and applies analysis on the captured data. The objective of this stage is to detect ownership change of the IoT device based on changes of its context. IoT devices are usually

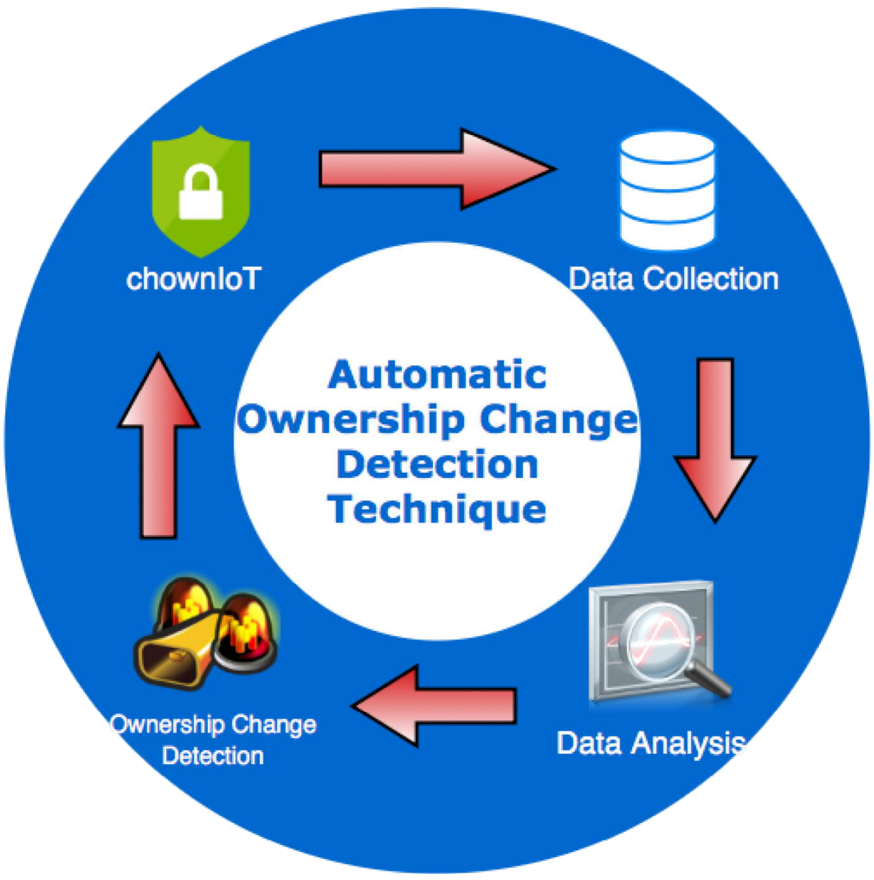
chownIoT
Data Collection
Automatic Ownership Change Detection Technique
Data Analysis
Ownership Change Detection

**Figure 4.1.** *FoundIoT* System Overview

deployed in a single physical location while they belong to a specific owner. The IoT device context characterizes its location. If the IoT device is moved to another location, its context will change. Therefore, the device context can be used to detect change of its location, which in most cases means change of its ownership.

If the IoT device is moved to a new location while preserving the same owner, the device owner will be at this new location and his other devices will likely be there too, which means that the IoT device context will not change.

For each captured context, *FoundIoT* computes the value of a metric that characterizes changes of the IoT device context. In case there is no ownership change, the value of the metric stays above the *detection threshold*. Decision on ownership change is made by detecting a continuous interval of metric values, lower than the detection threshold. If ownership change is detected, *FoundIoT* sends a signal to chownIoT to manage ownership change.

*FoundIoT* provides a multistage ownership change detection technique depicted in Figure 4.2. The first phase consists in detecting contextual changes based on Wireless Station (STA) in the device vicinity. If no change is detected, *FoundIoT* starts the AP detection procedure. This provides information about the set of active APs that announce their operation to STAs in the IoT device vicinity. The final phase checks for changes in context among Bluetooth enabled devices.

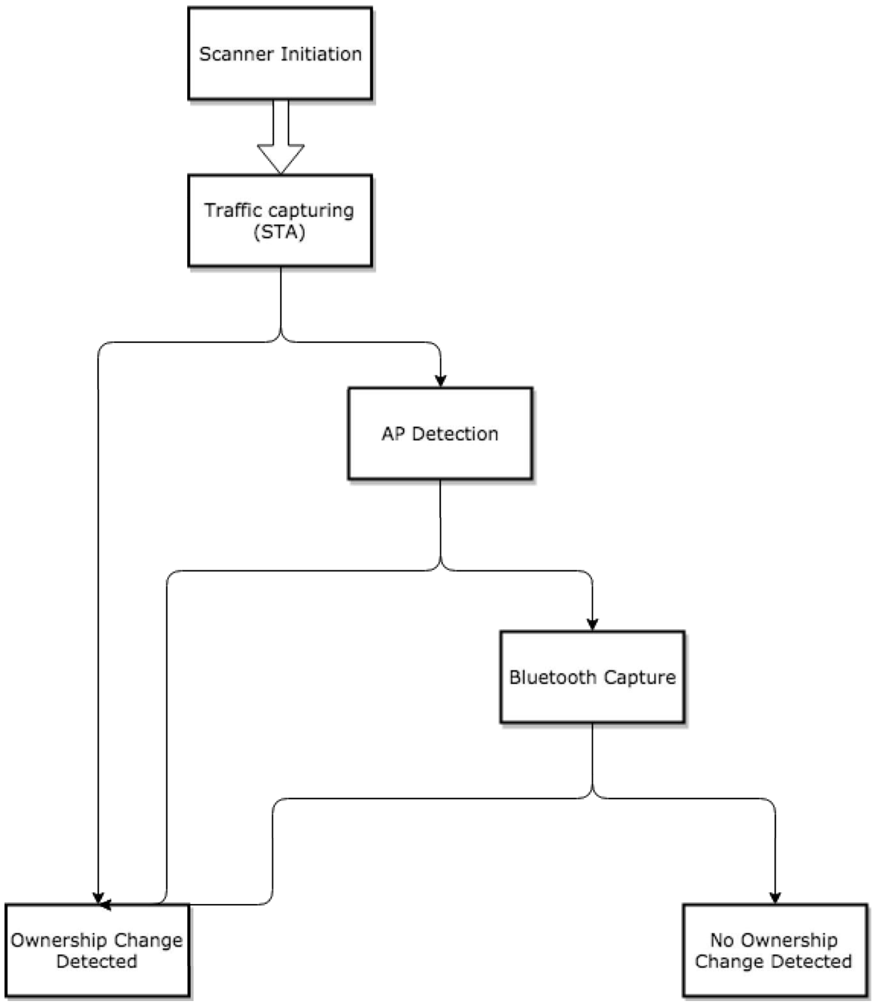

**Figure 4.2.** *FoundIoT* Ownership Change Detection Flow Diagram

## 4.2 Design Choices

The detection of ownership change can be achieved with various tools and techniques. This section presents the technologies and techniques chosen in *FoundIoT* implementation.

### 4.2.1 Context Identification

Smart home devices are equipped with various sensors, which can be used for scanning their environment. Smart thermostats use temperature and humidity sensors, smart home security systems provide cameras and proximity sensors.

The proposed solution is intended to support a wide range of smart home devices. Because of that, we cannot rely on uncommon sensor modalities and technologies for our implementation. This design choice is dictated by requirement **R1**, which is stated in Section 3.3.

To capture the context, the device needs to scan the network to detect connected devices in its vicinity. In general, IoT devices support diverse communication technologies, which could be used for establishing a connection and detecting device context. Some of the most widespread communication protocols are described in Section 2.2.

We base our solution on WiFi and Bluetooth for context identification and ownership change detection. This provides support of wide range of IoT devices. The choice to limit scanning techniques with these two specific communication channels involves a trade-off between requirements **R1** and **R3**.

### 4.2.2 Data recording

We store the collected data in a time series format. Time series implies having data with timestamps for each observation. This format is useful for analysis of processes that change over time.

When change of ownership happens, the system detects changes in the device context. It is important to identify specific points in time when the changes take place. Time series format can provide the system with required information, such as the timestamp of ownership change happening. Thus, time series format is beneficial for detection of ownership change.

### 4.2.3 Multistage Ownership Change Detection

*FoundIoT* provides a multistage protection scheme. This feature is implemented for mitigation of adversary attacks to disrupt the system detection method. It is rather easy for an adversary to spoof one of available ownership change detection methods. That is why detecting changes, using one specific method, is unreliable to make the decision on possible ownership change.

Figure 4.2 describes the multistage structure of *FoundIoT*. The system uses three different sequential checks to detect ownership change. Therefore, it increases *FoundIoT* resilience to adversary actions and provides more reliable assessment of contextual changes. This choice is justified by requirement **R3.1**, declared in Section 3.3.

### 4.2.4 Statistical analysis

Device contextual changes are analyzed using statistical techniques. Statistics helps improve reliability of the technique, decreasing the number of false alarms and mitigating adversary attempts to disrupt the system detection methods.

This design choice helps to achieve requirement **R4**.

## 4.3 Components

*FoundIoT* requires 4 components depicted in Figure 4.3.

**WiFi context monitor**

This component captures MAC addresses of STAs and APs in the device vicinity. Its design is described in Section 4.3.1. WiFi context monitor component generates a Tcpdump[7] pool of data, which is passed over to Time Series Generator.

**Bluetooth context monitor**

This component captures MAC addresses of Bluetooth enabled devices in device vicinity. Its design is described in Section 4.3.2. The list of captured MAC addresses is passed over to Time Series Generator.

**Time Series Generator**

Time Series Generator component prepares the data for its analysis. Its design is described in Section 4.3.3. The data gets converted into time series format with timestamped labels and passed over to Ownership Change Detection component.

**Ownership Change Detection**

The final component analyzes the time series data and detects ownership change of the IoT device. Its design is described in Section 5. The analysis is performed for each type of captured context: WiFi (STA and AP) and Bluetooth. Finally, the results are sent over to chownIoT for ownership change management.

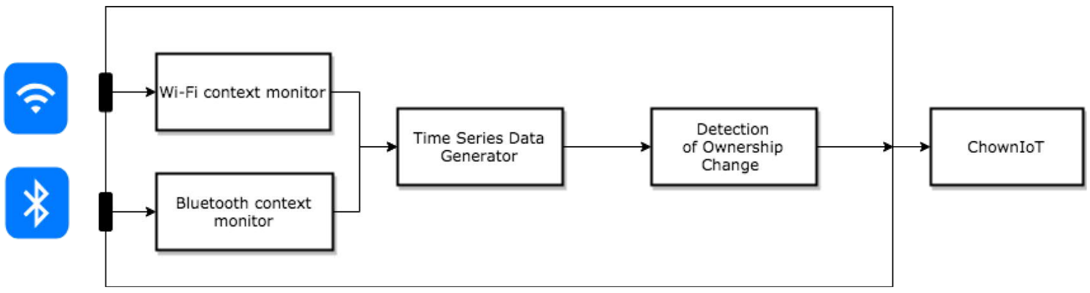
Wi-Fi context monitor
Bluetooth context monitor
Time Series Data Generator
Detection of Ownership Change
ChownIoT

**Figure 4.3.** *FoundIoT* Components

### 4.3.1 WiFi context monitor

The initial steps of WiFi context monitor are focused on scanning the device vicinity and capturing wireless traffic of STAs. The component performs two types of scanning procedures. At first, WiFi context monitor component scans the local network in promiscuous mode. This scan identifies the MAC addresses of STAs that operate in the same WiFi network as the IoT device.

Afterwards, the component makes another scan in monitor mode. It captures wireless traffic and identify MAC addresses of STAs in the IoT device vicinity by detecting network frames sent and transmitted by STAs on a specific frequency. That is why WiFi context monitor component dynamically changes frequency of the network card adapter to capture packets from every STA in the IoT device vicinity. The set of frequencies is limited by the wireless channels supported by the network card manufacturer.

Scans performed on each wireless frequency provide different insight on current context. That is why the component performs each scan with distinct duration. Different values of scan duration and their impact on detection accuracy and system performance are evaluated in Chapter 7.

AP detection is performed via Beacon capturing. Beacon is a management frame in IEEE 802.11 standard [6]. APs transmit these frames periodically to announce their service to STAs. Beacons frames are extracted from monitor mode scan results.

### 4.3.2 Bluetooth context monitor

Bluetooth context monitor component performs scans using standard Bluetooth discovery features of the IoT device. It launches *Bluetoothctl* tool [18] to scan for Bluetooth enabled devices in the vicinity.

Bluetoothctl provides basic mechanisms of Bluetooth pairing, establishing connection between devices. It lists the devices that are discovered in the device vicinity by recording their MAC addresses. Bluetooth context monitor component stores the discovered MAC addresses with a timestamp of their discovery.

### 4.3.3 Time Series Generator

After the wireless activity capture procedure is complete, the system receives unformatted data about captured WiFi and Bluetooth devices. It needs to be transformed into a specific format to apply required analysis techniques.

*FoundIoT* transforms the captured stream of data to time series format. Each observation is assigned with a unique identifier, the timestamp of when it was discovered. This allows to put each observation on the timeline, analyze the resulting sequence and extract its features.

$$\begin{gathered}\{TS_1 : [\{TS_{1.1}, MAC_{1.1}\} \dots \{TS_{1.m}, MAC_{1.m}\}],\\ TS_2 : [\{TS_{2.1}, MAC_{2.1}\} \dots \{TS_{2.k}, MAC_{2.k}\}],\\ \vdots\\ TS_N : [\{TS_{N.1}, MAC_{N.1}\} \dots \{TS_{N.l}, MAC_{N.l}\}]\}\end{gathered} \tag{4.1}$$

The data is formatted according to the format depicted in Equation 4.1. Each observation consists of several pairs of device MAC address and the timestamp of its discovery.

As a result, Time Series Generator component creates 3 time series: $TS_{STA}$, $TS_{AP}$, and $TS_{Bluetooth}$ for discovered STAs, APs and Bluetooth devices.

# 5. Ownership Change Detection

This chapter specifies the chosen techniques and algorithms for the analysis of collected time series data structure. The output from these methods is used for identifying points in time of possible device ownership change.

## 5.1 Time Series Analysis

Ownership Change Detection component receives 3 time series from Time Series Generator component, described in Section 4.3.3. The first time series $TS_{STA}$ contains information about active STAs. We differentiate local and external devices. We define local devices as those which are connected to the same (local) WiFi network as the IoT device. Therefore, external devices do not belong to the local network and are connected to other APs. Each examined observation consists of two subsets: local and external devices, which are generated by promiscuous and monitor scanning procedures, respectively.

The second time series $TS_{AP}$ contains information about nearby APs. As for $TS_{STA}$ time series, $TS_{AP}$ contains two types of devices: local and external APs. The local AP is defined as the AP to which the IoT device is connected to. All other discovered APs are defined as external ones.

The third time series $TS_{Bluetooth}$ contains information about Bluetooth devices in the IoT device vicinity.

All mentioned time series are analyzed, using a metric that evaluates similarity of observations within each time series. To evaluate similarity of time series, we need to choose an appropriate metric for our data. In the following sections 5.1.1 and 5.1.2, we describe two metrics that can be used for similarity evaluation of our time series.

### 5.1.1 Jaccard Index

Jaccard Index [19] compares members of two sets to identify shared and distinct members. It is a measure of similarity for two datasets in range [0, 1]. The higher the value is, the more similar analyzed datasets are.

Several prior research works [20], [21] used Jaccard Index previously for wireless context comparison. Jaccard Index is effective in calculating similarity of data objects that have binary attributes, it can be applied to our collected data. Each *FoundIoT* scan observation consists of the set of detected wireless devices. A device could either be detected or not due to its wireless inactivity.

Jaccard Index provides proper results for sets of different cardinality. It is achieved by normalizing the score by union of analyzed sets. Without this normalization, small sets would always have very low scores, which negatively affects the accuracy of similarity computations.

Jaccard Index of two sets A and B is expressed as:

$$JaccardIndex(A, B) = \frac{|A \bigcap B|}{|A \bigcup B|} \tag{5.1}$$

*FoundIoT* performs the analysis of the collected contexts periodically, each 10 minutes. Each scan is assigned with the timestamp $scan_{timestamp}$ of when it was performed.

On each analysis iteration, the value of Jaccard Index gets computed based on the sets of all previous and the current scans. The system composes the union of MAC addresses of devices discovered prior to the current scan. The resulted set, $Set_{prev}$, consists of unique MAC addresses of devices that were discovered during $[scan_{timestamp_1} ... scan_{timestamp_{N-1}}]$ scanning procedures.

Next, the system computes the intersection and union of $Set_{prev}$ with the set $Set_{curr}$ that contains MAC addresses of devices discovered during the current scan $scan_{timestamp_N}$. The Jaccard Index metric value for the current scan is computed, according to the following equation:

$$\begin{aligned} JaccardIndex_{scan_{timestamp_N}} &= \frac{|Set_{prev} \bigcap Set_{curr}|}{|Set_{prev} \bigcup Set_{curr}|} = \\ &= \frac{|\bigcup_{i=timestamp_1}^{timestamp_{N-1}} scan_i \bigcap scan_{timestamp_N}|}{|\bigcup_{i=timestamp_1}^{timestamp_{N-1}} scan_i \bigcup scan_{timestamp_N}|} \end{aligned} \tag{5.2}$$

*FoundIoT* computes the Jaccard Index value for each observation within received datasets. Computed values are added to $TS_{JaccardIndex}$ that represent the similarity of $TS_{STA},\ TS_{AP},\ and\ TS_{Bluetooth}$.

### 5.1.2 Kullback-Leibler divergence

The Kullback-Leibler divergence [22], [23] is a measure of how two probability distributions diverge. Kullback-Leibler divergence gives a value in the range [0, 1]. The greater the value is, the more divergent distributions are. Low values of Kullback-Leibler divergence indicate similar distributions.

The Kullback-Leibler divergence is used in a variety of fields, including mathematical statistics, evaluating relative entropy in information systems, randomness and similarity in time series sequences. Lee et al. [24] used KL Divergence metric for time series clustering, and Liu [25] used it for change-point detection in time series.

Kullback-Leibler divergence of two distributions P and Q is defined by Equation 5.3.

$$D_{KL}(P||Q) = \sum_{i=1}^{N} P(i) * \log \frac{P_i}{Q_i} \tag{5.3}$$

The value of Kullback-Leibler divergence gets computed based on the same two datasets, as the Jaccard index: $Set_{prev}$, which consists of MAC addresses of all discovered devices during $[scan_{timestamp_1}...scan_{timestamp_{N-1}}]$, and $Set_{curr}$ with MAC addresses from current observation.

During each analysis iteration, ***FoundIoT*** computes the number of occurrences of each MAC address from the analyzed scan period $[scan_{timestamp_1}...scan_{timestamp_{N-1}}]$. To compute the number of device occurrences, we define a function $f(MAC_{device}, Set_{MAC})$ in Equation 5.4, which takes two parameters. The first parameter is MAC address of the analyzed device. The second parameter is a set of MAC addresses $Set_{MAC}$.

The function returns 1 if $MAC_{device}$ is present in $Set_{MAC}$. In other cases, the function returns 0.

$$f(MAC_{device}, Set_{MAC}) = \begin{cases} 1, & \text{if } MAC_{device} \in Set_{MAC} \\ 0, & \text{else} \end{cases} \tag{5.4}$$

During each $scan_{timestamp_N}$, ***FoundIoT*** computes the sum of values of the function defined in Equation 5.4 for each device from $Set_{prev}$. The resulted value is divided by the number of scans $N-1$ to compute the device occurrence in $Set_{prev}$.

The occurrence value is used to assign weight $Weight_{MAC_{device}}$ for each discovered device from $Set_{prev}$:

$$Weight_{MAC_{device}} = \frac{\sum\limits_{i=1}^{N-1} f(MAC_{device}, Set_{MAC_i})}{N-1} \tag{5.5}$$

The first step of computing divergence between sets $Set_{prev}$ and $Set_{curr}$ is the computation of occurrence weights $Weight_{occurrence}$ for each device from the intersection of the sets (Equation 5.6).

$$Weight_{occurrence} = \frac{Weight_{MAC_{device}}}{\sum_{i=1}^{M} Weight_{MAC_{device_i}}}, \tag{5.6}$$

where M is the size of intersection $|Set_{prev} \bigcap Set_{curr}|$.

We assign occurrence weights $Weight_{occurrence}$ for each MAC address from the intersection $Set_{prev} \bigcap Set_{curr}$ based on its occurrence in $Set_{prev}$ and normalized weights $Weight_{normalized} = \frac{1}{|Set_{prev} \bigcap Set_{curr}|}$ based on the size of the intersection.

The equation for Kullback-Leibler divergence for *FoundIoT* has the following form:

$$D_{KL}(Set_{prev}||Set_{curr})_{scan_{timestamp_N}} = \sum_{i=1}^{M} Weight_{occurrence_i} * \log \frac{Weight_{occurrence_i}}{Weight_{normalized}} \tag{5.7}$$

We are using both described metrics for similarity evaluation of the IoT device context to compare *FoundIoT* detection performance with each of them.

The main advantage of Kullback-Leibler divergence over Jaccard Index metric is its ability to assign weights for each device, based on its occurrence. The higher the device occurrence is, the greater its impact on the divergence value is. As a result, devices that belong to a different context should significantly decrease the similarity score. KL Divergence is supposed to provide more accurate results and fit any scenario. However, it will require more computations to compute the occurrence of each device compared to Jaccard Index, which does not take into account the occurrence of devices.

On the contrary, Jaccard Index metric assigns equal weights to discovered devices and computes the similarity score with lower number of computations. It is supposed to provide more accurate results in case of contexts with devices with stable occurrence. Therefore, both metrics has their advantages and disadvantages due to their technique of the similarity score computation.

Chapter 7 provides a detailed evaluation of the impact of each similarity metric on *FoundIoT* performance.

## 5.2 Captured Data Filtering

STAs do not use wireless communication channels constantly. The connection is established per request from the device and stays open until the AP forces it to renew the Dynamic Host Configuration Protocol (DHCP) lease.

It means that some STAs will not be present in the analyzed observation due to their network inactivity or outdated DHCP lease. Such devices cause misleading decreases of the similarity score.

That is why, *FoundIoT* sets the ignore limit $\Delta_{Ignore}$ for STAs with low network activity. These could be stationary devices, which require occasional network activity for their operation or portable devices that are present in the network for a short-term time. These devices get excluded from the analysis to reduce the number of false alarms.

During each $scan_{timestamp_N}$, *FoundIoT* computes the number of occurrences of each device from period $[scan_{timestamp_1}...scan_{timestamp_{N-1}}]$. If it is lower than $\Delta_{ignore}$, then it gets ignored during the current analysis iteration. Afterwards, *FoundIoT* computes the sum of values of function depicted in Equation 5.4 for each $MAC_{device}$ from $Set_{prev}$. The resulted value is divided by the number of scans $N-1$ to compute the occurrence of $MAC_{device}$ in $Set_{prev}$, which is depicted in Equation 5.8.

$$\frac{\sum_{i=1}^{N-1} f(MAC_{device}, observation_i)}{N-1} <= \Delta_{ignore} \tag{5.8}$$

The parameter $\Delta_{ignore}$ sets a lower bound for device occurrence. The higher it is, the more devices with low network activity get excluded from analysis during the current iteration.

## 5.3 Analysis Conclusions

Ownership Change Detection component studies the evolution of similarity metrics changes over consecutive scans. It detects decreases of Jaccard Index values and increases of KL divergence, which correspond to lower similarity. When the system detects a long-term continuous interval of Jaccard Index similarity scores lower than $\Delta_{JaccardIndex}$, it indicates a possible ownership change situation. For KL divergence, we need to detect a long-term continuous interval with similarity scores greater than $\Delta_{KL\ divergence}$.

To detect a continuous interval of low Jaccard Index similarity scores, we define a function $f(Score_{JaccardIndex}, \Delta_{JaccardIndex})$ in Equation 5.9. It returns 1 if the passed Jaccard Index similarity score is lower than the detection threshold $\Delta_{JaccardIndex}$. In other cases, it returns 0.

$$f(Score_{JaccardIndex}, \Delta_{JaccardIndex}) = \begin{cases} 1, & \text{if } Score_{JaccardIndex} < \Delta_{JaccardIndex} \\ 0, & \text{else} \end{cases} \tag{5.9}$$

To detect a continuous interval of high KL divergence similarity scores, we define another function $f(Score_{KL\ Divergence}, \Delta_{KL\ Divergence})$ in Equation 5.10. It returns 1 if the passed KL divergence similarity score is greater than the detection threshold $\Delta_{KL\ Divergence}$. In other cases, it returns 0.

$$f(Score_{KL\ Divergence}, \Delta_{KL\ Divergence}) = \begin{cases} 1, & \text{if } Score_{KL\ Divergence} > \Delta_{KL\ Divergence} \\ 0, & \text{else} \end{cases} \tag{5.10}$$

Ownership Change Detection condition is depicted in Equation 5.11. *FoundIoT* detects change of ownership when the number of consecutive similarity scores exceeds the window $W$.

$$\sum_{i=1}^{N} f(Score_{similarity_i}, \Delta_{similarity}) >= W \tag{5.11}$$

The system consecutively analyzes each TS: $TS_{STA}$, $TS_{AP}$ and $TS_{Bluetooth}$, and checks if Ownership Change Detection condition is respected. By analyzing $TS_{STA}$ and $TS_{AP}$, *FoundIoT* checks for changes in local and external WiFi contexts captured in each TS. By analyzing $TS_{Bluetooth}$, *FoundIoT* checks for changes in Bluetooth context. In case the Ownership Change Detection condition is respected during the analysis of one of TS, the system indicates the detection of ownership change and does not check the remaining TS.

The parameters $\Delta_{similarity}$ (Jaccard Index or KL Divergence metric) and $W$ affect the system detection accuracy. $\Delta_{similarity}$ sets the detection threshold for similarity score that indicates changes in device context, and therefore ownership change of the IoT device.

$W$ sets a window of low similarity scores used to filter out temporary decreases in similarities between the observations. Thus, these parameters affect the number of false alarms and detection sensitivity.

**False Positive (FP)**

A false alarm or false positive is an event, where *FoundIoT* detects ownership change while it did not actually happen;

**False Negative (FN)**

A false negative is an event, where *FoundIoT* does not detect ownership change while it actually happened;

**True Positive (TP)**

A true positive is an event, where *FoundIoT* detects ownership change while it actually happened;

**True Negative (TN)**

A true negative is an event, where *FoundIoT* does not detect ownership change while it did not actually happen.

The false positive rate is computed according to Equation 5.12, which presents the ratio between the number of negative events incorrectly identified as positive and the total number of actual negative events.

$$FPR = \frac{FP}{FP + TN} \tag{5.12}$$

Detection sensitivity (also called the true positive rate) measures the proportion of positives that are correctly identified as such. This metric evaluates the *FoundIoT* efficiency. It gives the score of how many actual ownership change events were detected correctly. The formula for true positive rate is presented in Equation 5.13.

$$TPR = \frac{TP}{TP + FN} \tag{5.13}$$

# 6. Implementation

In chapters 4 and 5, we have provided a detailed description of the design choices and components of the proposed solution. This chapter provides an overview of *FoundIoT* prototype implementation.

As it was mentioned in Section 4.3, *FoundIoT* requires WiFi context monitor, Bluetooth context monitor, Time Series Generator and Ownership Change Detection components. Therefore, we need to implement WiFi and Bluetooth context monitoring, time series generation and analysis techniques. Apart from these, we also need a platform for implementing the smart home device functionality.

The following sections provide the implementation description for each *FoundIoT* component and the prototype IoT device.

## 6.1 Smart Home Device

IoT devices are designed to implement various functionality. Because of that, vendors provide diverse software and hardware specifications and capabilities to manufactured IoT devices. In addition, most vendors do not provide open source programming specifications with IoT devices due to business requirements.

To implement a prototype of *FoundIoT*, we need an IoT testbed device. We have chosen Raspberry Pi 3 [11] development board running on Raspbian Jessie OS [26] for our implementation as it represents a typical IoT device:

- It provides similar capabilities (WiFi and Bluetooth connectivity);
- It is a low cost device as most off-the-shelf IoT devices.

In addition, it supports a variety of modern programming languages, including Python and command-line tools, such as Tcpdump [7] and Bluetoothctl, which are used for the prototype implementation of *FoundIoT*.

## 6.2 WiFi Context Monitoring

*FoundIoT* uses onboard WiFi chip capabilities to scan the IoT device vicinity and monitor the WiFi context. One of the most popular traffic monitoring tools is Tcpdump. It is widely supported in modern operating systems and provides required functionality for capturing STAs wireless traffic. *FoundIoT* launches the Tcpdump capture, monitors wireless activity and saves the collected network trace to a *.pcap* [27] file. The resulted file contains network frames, including source and destination MAC addresses of communicating nodes, timestamps and encrypted data. Next, *FoundIoT* filters out source MAC addresses on outgoing network frames and timestamps from the trace and saves them to a *.txt* file.

To capture APs in the IoT device vicinity, we use Linux standard *iwlist* [28] utility. It invokes the device standard WiFi discovery mechanisms and prints out all available nearby APs. *FoundIoT* extracts the list of MAC addresses of nearby APs from iwlist output and saves it with the timestamp to a .txt file.

## 6.3 Bluetooth Context Monitoring

To capture the Bluetooth context, *FoundIoT* uses standard Bluetooth discovery tools of the device. In our case, Bluetoothctl tool represents base functionality of the IoT device Bluetooth discovery mode. Bluetoothctl provides the list of nearby Bluetooth active devices and can be used on a variety of modern Linux distributions.

*FoundIoT* launches Bluetoothctl to capture MAC addresses of Bluetooth devices in the IoT device vicinity and saves the list of their MAC addresses to a .txt file.

## 6.4 Time Series Generation

After *FoundIoT* captures WiFi and Bluetooth contexts, it needs to format the data to time series for analysis.

We use Python programming language for context formatting. It provides a variety of useful tools and libraries for transforming raw context data into JavaScript Object Notation (JSON) structure that represents time series in our case.

We implement time series generation in Python using JSON library [29]. We use collected WiFi and Bluetooth raw contexts with timestamps as the input parameters. JSON library formats the collected contexts and generates a JSON structure where timestamps represent keys of records and collected contexts - records. The resulted structure represents collected WiFi and Bluetooth contexts in a time series format.

## 6.5 Time Series Analysis

The analysis of time series is performed using Python Scipy [30] and Scikit-learn [31] libraries. These libraries provide implementations of Jaccard Index and KL Divergence metrics used by *FoundIoT* for context similarity evaluation.

*FoundIoT* invokes *jaccard_similarity_score* function, which is a part of Python Scikit-learn library during each scan iteration and passes sets of previously discovered and current MAC addresses as input parameters to it. The value provided by *jaccard_similarity_score* function represents the Jaccard Index similarity score of sets of MAC addresses.

The computation of KL Divergence metric value is provided by Scipy library. *FoundIoT* invokes *scipy.stats.entropy* function that calculates the entropy of a distribution for given probability values. *FoundIoT* provides occurrence and normalized weights of MAC addresses as input parameters to the entropy function. The resulted value represents the KL Divergence score of sets of previously discovered and current MAC addresses.

# 7. Evaluation

The chapter evaluates the described system features and components based on the identified requirements in Section 3.3. We evaluate deployability, resource constraints, accuracy, speed and security of *FoundIoT* in detail.

## 7.1 Deployability

In this section, we evaluate the deployability of *FoundIoT*.

***Software limitations***: *FoundIoT* uses C++, Python and shell scripting to implements the main features of ownership change detection. These software packages and technologies are widely supported by most modern operating systems. Thus, the proposed system can be deployed in any modern operating system.

***Hardware limitations***: *FoundIoT* does not depend on any particular hardware component. It uses wireless communication channels, such as WiFi and Bluetooth, for its operation. It can be implemented on a wide range of devices that support wireless communication, which is a required component for a smart home device.

***Sensor limitations***: *FoundIoT* uses wireless adapters for detection of ownership change. Currently, the system uses WiFi and Bluetooth adapters. The set of supported adapters can be easily expanded for improving the accuracy of ownership change detection on a specific device. Thus, the system can be deployed on devices with any number of sensor modalities.

As a result, the implemented prototype of *FoundIoT* can be deployed on a wide range of IoT devices, as it is based on widely supported software and hardware solutions, and it does not depend on sensor modalities that are specific for a limited number of IoT devices. The requirement **R1** is fully met.

## 7.2 Resource Constraints

In this section, we evaluate the resource usage of *FoundIoT* on Raspberry Pi 3.

Usually, smart home devices are designed to perform a limited set of simple actions. That is why, device manufacturers equip them with limited processing power (CPU) and memory (RAM). To be properly supported by resource constrained devices, we need to ensure that our solution operates efficiently and does not generate too much overload. In this section, we evaluate the performance of *FoundIoT* execution.

Some popular IoT boards are presented in Table 7.1. As we can see, all boards provide a limited amount of resources. Raspberry Pi 3 is the most computationally powerful device with 1.2 GHz CPU and 1 GB of RAM. In contrast, Tessel 2 is the least powerful device among presented. It is equipped with only 580 GHz CPU and 64 MB of RAM.

| **Board** | **CPU** | **RAM** | **Disk** |
|---|---|---|---|
| Tessel 2 [16] | MediaTek MT7620n 580 MHz | 64 MB | 32 MB |
| C.H.I.P [32] | Allwinner R8 1 GHz | 512 MB | 4 GB |
| Intel Edison [33] | Atom Silvermont 500 MHz | 1 GB | 4 GB |
| Udoo Neo [34] | ARM Cortex A9 1 GHz | 1 GB | up to 256 GB |
| Raspberry Pi 3 [11] | ARM Cortex A53 1.2GHz | 1 GB | up to 256 GB |

**Table 7.1.** Specifications of IoT Boards

The most resource consuming components of *FoundIoT* are WiFi and Bluetooth context monitors and Ownership Change Detection. The amount of data stored to analyze by Ownership Change Detection component depends on the number of discovered devices by WiFi and Bluetooth context monitor components. The captured data that is used for evaluation was recorded in environments with different number of wireless devices. Thus, we evaluate the central processor and memory usage required by *FoundIoT*.

The experiment setup consists of three main parts: the Raspberry Pi 3 as an IoT testbed device, implementation of *FoundIoT* on the testbed device and the chosen environment.

We evaluate CPU and RAM usage of traffic capturing component of *FoundIoT* in a crowded environment with a large number of active wireless devices in the device vicinity (Computer Science Department building of Aalto University). In general, smart home devices are located in less crowded environments with a lower number of devices.

We present resource usage of *FoundIoT* only in a very crowded environment to show an extreme case of high system load on the IoT device. During one week of periodic scans, the system detected 713 STAs, 77 APs and 604 Bluetooth devices. Other environments will show approximately the same or lower load.

We measure CPU and RAM usage of WiFi and Bluetooth context monitors component periodically, every 2 minutes during execution, using ps utility [35]. Table

7.2 presents the acquired measurements for monitor components of *FoundIoT*.

| | WiFi mon. | | Bluetooth mon. |
|---|---|---|---|
| | **STA** | **AP** | **Bluetooth** |
| CPU usage, % | $0.05\% \pm 0.01\%$ | $0.05\% \pm 0.01\%$ | $1.86\% \pm 0.23\%$ |
| RAM usage, % | $0.28\% \pm 0.04\%$ | $0.18\% \pm 0.02\%$ | $0.89\% \pm 0.03\%$ |
| RAM average usage, MB | 2.6 MB | 1.6 MB | 8.3 MB |

**Table 7.2.** CPU and RAM usage of *FoundIoT* monitor components on the Raspberry Pi 3

Another component of *FoundIoT*, which needs to be evaluated, is Ownership Change Detection. We evaluate CPU and RAM usage in the same crowded environment where data capture was measured. Table 7.3 presents the acquired measurements for the Ownership Change Detection component of *FoundIoT*.

| | WiFi con. analysis | | Bluetooth con. analysis |
|---|---|---|---|
| | **STA** | **AP** | **Bluetooth** |
| CPU av. usage, % | $0.38\% \pm 0.06\%$ | $0.35\% \pm 0.02\%$ | $0.43\% \pm 0.05\%$ |
| RAM usage, % | $0.67\% \pm 0.05\%$ | $0.54\% \pm 0.03\%$ | $0.48\% \pm 0.02\%$ |
| RAM av. usage, MB | 6.2 MB | 4.9 MB | 4.4 MB |

**Table 7.3.** CPU and RAM usage of *FoundIoT* Ownership Change Detection component on the Raspberry Pi 3

As a result, the maximum total memory consumption of *FoundIoT* is 28 MB.

Figure 7.1 presents persistent storage usage of periodic network scans performed by *FoundIoT* during one week in the same location, Aalto CS building. As we can see from the figure, the system requires approximately 1 MB of persistent storage to store the data captured during a week of scans. *FoundIoT* disk space requirements are compliant with presented IoT boards in Table 7.1.

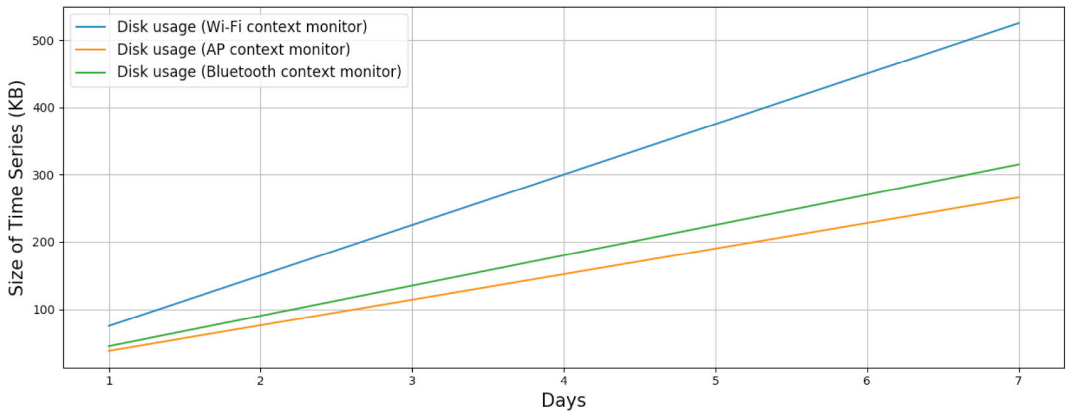

**Figure 7.1.** *FoundIoT* persistent storage usage over 1 week of scans

Thus, the minimum specification for supporting *FoundIoT* is 32 MB of RAM and 16 MB of persistent storage. The total *FoundIoT* resource usage does not create a significant overhead to any of IoT boards presented in Table 7.1. It can be integrated to a wide range of IoT devices without consuming a noticeable amount of resources and creating an overhead.

As a result, we can state that the prototype implementation of *FoundIoT* can operate on IoT devices with limited resources and does not cause high system load. Requirement **R2** is fully met.

## 7.3 Performance

In this section, we evaluate the system performance by addressing two requirements. First, we determine ownership change scenarios and evaluate the accuracy requirement **R3.1** by computing true positive and false positive rates to find optimal detection parameters. Second, we evaluate the speed requirement **R3.2** by comparing speed of detection with Jaccard Index and KL Divergence metrics.

### 7.3.1 Ownership Change Scenarios Description

In this section, we present different scenarios that need to be supported to detect ownership change accurately. We used different IoT devices for testing *FoundIoT* detection accuracy in ownership change scenarios. Table 7.4 presents IoT devices and their identifiers used for scenario implementation.

| Testing environment | IoT device | Identifier |
|---|---|---|
| 1 | Netatmo Thermostat #1 | IoT 1.1 |
| | Xiaomi Gateway #1 | IoT 1.2 |
| | Mobile Alerts Weather Station #1 | IoT 1.3 |
| | eWeLink Smart Socket #1 | IoT 1.4 |
| | Broadlink Environment Sensor #1 | IoT 1.5 |
| 2 | Netatmo Thermostat #2 | IoT 2.1 |
| | Xiaomi Gateway #2 | IoT 2.2 |
| | Mobile Alerts Weather Station #2 | IoT 2.3 |
| | eWeLink Smart Socket #2 | IoT 2.4 |
| | Broadlink Environment Sensor #2 | IoT 2.5 |

**Table 7.4.** IoT devices identifiers used in ownership change scenarios

Each figure that represents a scenario depicts two distinct environments with a local network (Local Network) and a set of external WiFi networks (Ext. Net.) that operate in the IoT device vicinity.

*FoundIoT* performs the analysis of three time series $TS_{STA}$, $TS_{AP}$ and $TS_{Bluetooth}$ to detect ownership change. $TS_{STA}$ and $TS_{AP}$ contain two subsets that represent local and external STA and AP contexts similarity. These subsets provide time series with Jaccard Index or KL Divergence similarity scores, which are evaluated to detect ownership change.

Next, we present the description of ownership change scenarios that *FoundIoT* needs to support for accurate detection of the IoT device ownership change.

**Scenario 1** Selling IoT device to a person that lives in a different house.

During the IoT device life cycle, it could be sold to a different person, a new owner, or it can be stolen by an adversary. Usually, a new owner lives in a different environment, which implies a completely different IoT device context. An example scheme of such scenario is depicted in Figure 7.2.

Figure 7.2 shows two testing environments without shared devices, which implies the IoT device relocation to a new owner's environment.

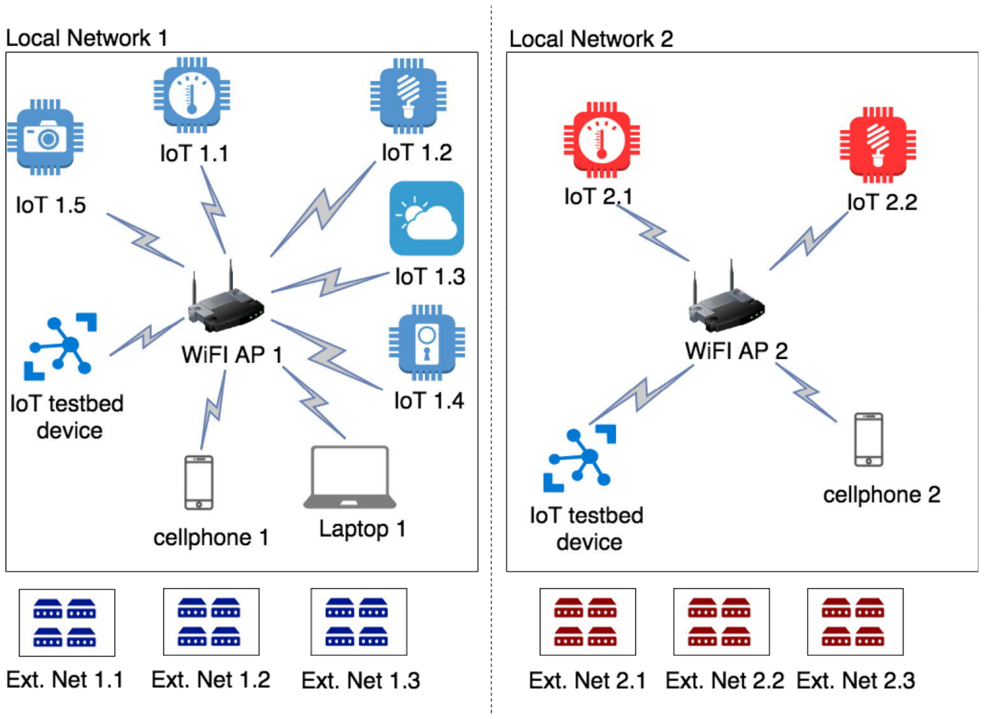

**Figure 7.2.** Scenario 1 - Selling device or the device theft (new owner in a different location).

**Scenario 2** Selling IoT device to a person that lives in vicinity of the previous owner (neighbor).

This scenario describes a less common IoT device selling situation. In this case, a new owner lives close to the previous owner, an adjacent apartment. An example scheme of the scenario is depicted in Figure 7.3.

Figure 7.3 shows two environments with shared devices in external STA and AP contexts due to their close location.

**Scenario 3** Selling smart home with an integrated IoT device.

Another ownership change scenario happens in case of selling a smart home with integrated IoT device. An example scheme is depicted in Figure 7.4. In the previously discussed scenarios, the IoT device is taken from one environment and placed in another. This scenario implies changes in context without changing the location of the IoT device itself.

Figure 7.4 shows two environments with shared devices in local contexts due

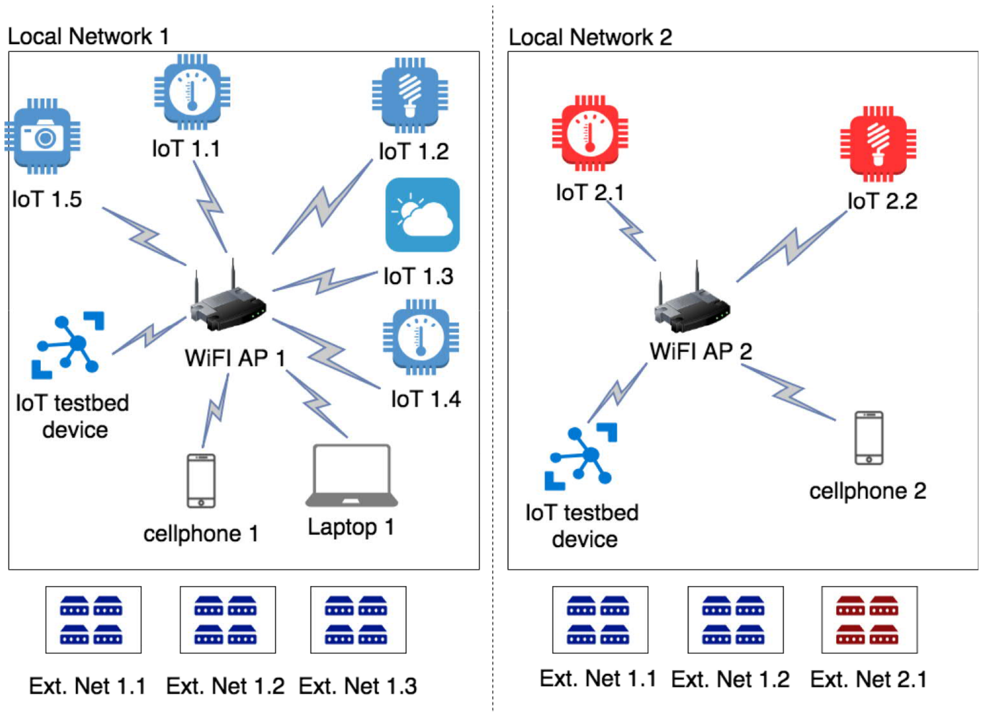

**Figure 7.3.** Scenario 2 - Selling IoT device to a neighbor.

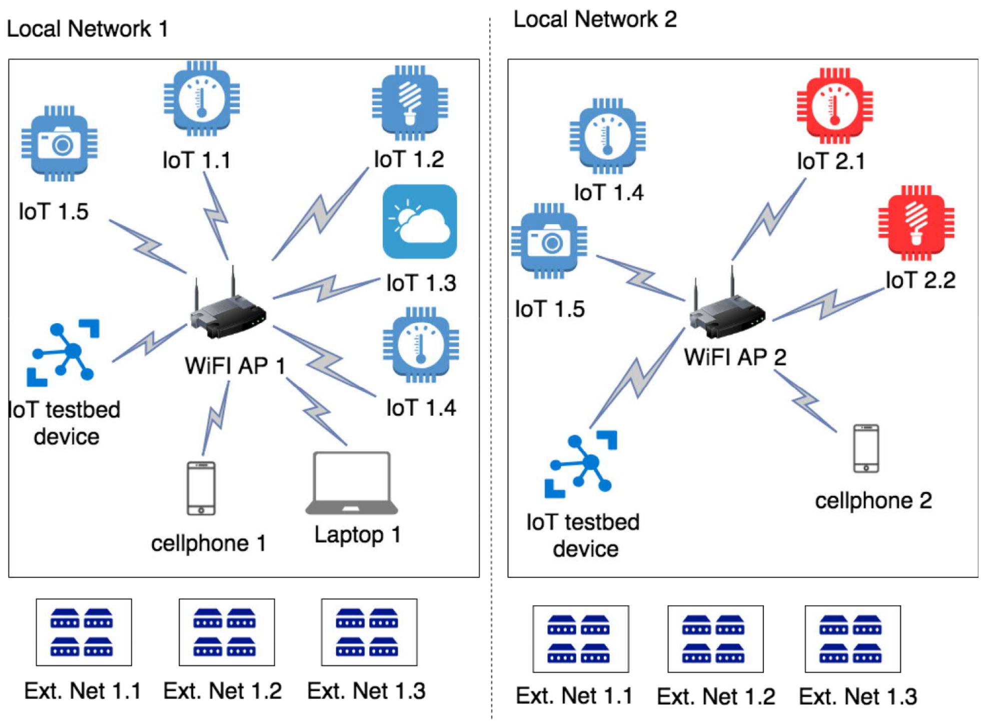

**Figure 7.4.** Scenario 3 - Selling smart home with integrated IoT device

to devices that were left by the previous owner and the external contexts because the location of the IoT device has not changed.

Next, we present Jaccard Index metric observations for each analyzed context in ownership change scenarios 1-3.

In the first scenario, *FoundIoT* does detect any previously discovered devices that represented the previous owner's context after the ownership change. A new owner has completely different devices that are connected to his WiFi network, which indicates significant contextual changes within the local network. External WiFi networks also have completely different devices, compared to the previous owner's external context.

Figures 7.5, 7.6 and 7.7 present the observation of Jaccard Index metric over one week of scans for Scenario 1. Figure 7.5 illustrates the observation for $TS_{STA}$ by showing a chart with two lines: blue - for representing similarity of local context, orange - in external context. Figure 7.5 shows a drop in similarity for local and external contexts during the 580th scan. The drop is caused by change of device location to a different environment. The ownership change happened when the similarity for both contexts experienced a significant drop.

Figures 7.6 and 7.7 illustrate the system analysis results for $TS_{AP}$ and $TS_{Bluetooth}$ using Jaccard Index metric. The metric can clearly distinguish two environments by showing continuous zero percent similarity in the transition area.

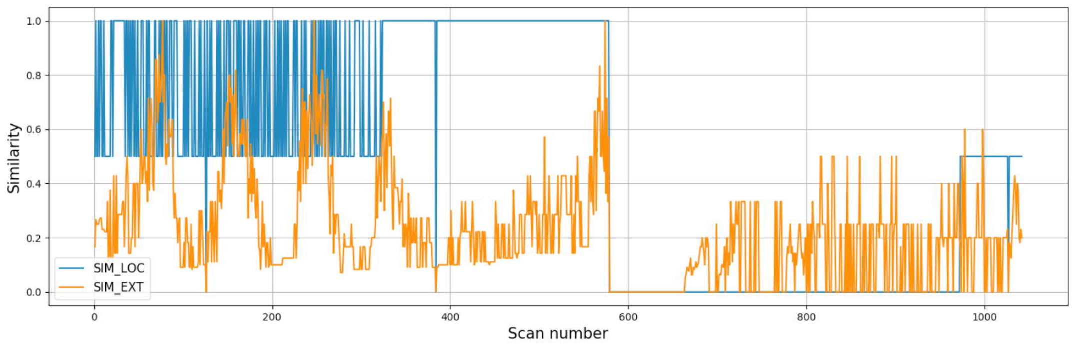

**Figure 7.5.** Scenario 1. $TS_{STA}$ observation by Jaccard Index metric (ownership changed during the 580th scan).

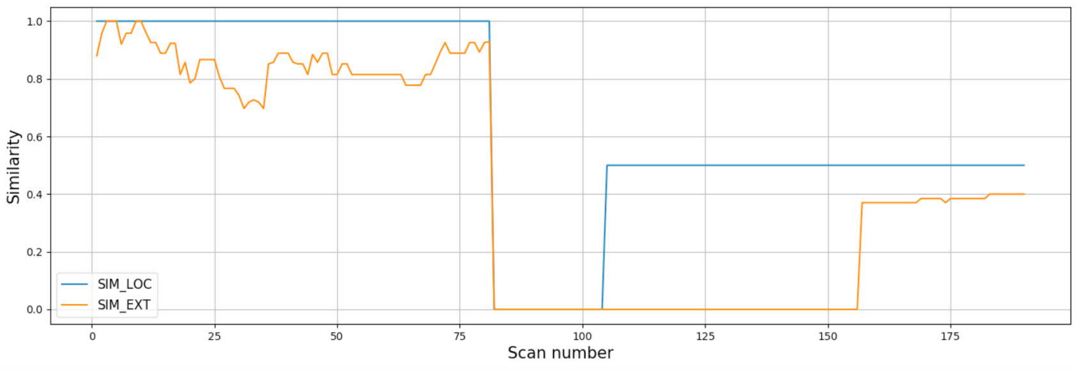

**Figure 7.6.** Scenario 1. $TS_{AP}$ observation by Jaccard Index metric (ownership changed during the 81st scan).

Figures 7.8, 7.9 and 7.10 present the observation of Jaccard Index metric over one week of scans in case of Scenario 2. As Figure 7.8 illustrates, the Jaccard Index metric can differ previous and new owner's local contexts by showing a significant drop during 580th scan.

However, the previous and new owner's external contexts have shared devices,

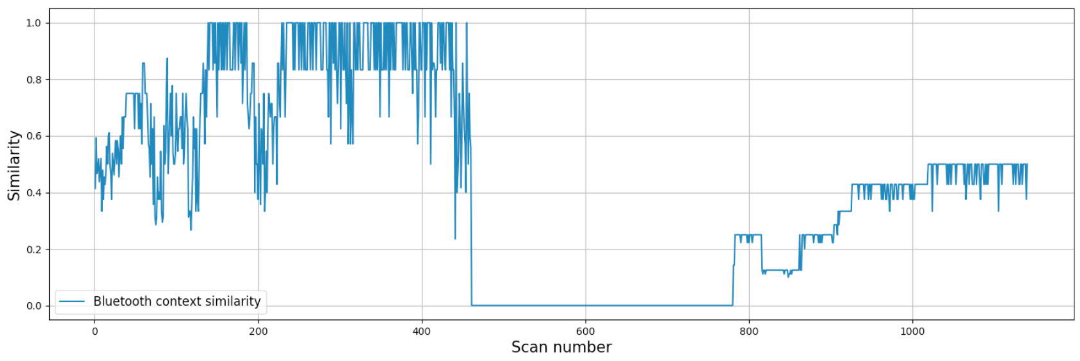

**Figure 7.7.** Scenario 1. $TS_{Bluetooth}$ observation by Jaccard Index metric (ownership changed during the 470th scan).

because of their close location. Jaccard Index metric does not show a significant drop in Figure 7.8, as there are previously discovered devices along with some new ones after ownership change.

Figures 7.8 - 7.10 show clear distinction in local contexts in two different environments. This distinction is caused by different devices that are connected to local networks in both environments. On the contrary, Jaccard Index metric analysis shows resemblance in external WiFi and Bluetooth contexts without significant drops in similarity. Similarity between these contexts is present because of close location of the environments.

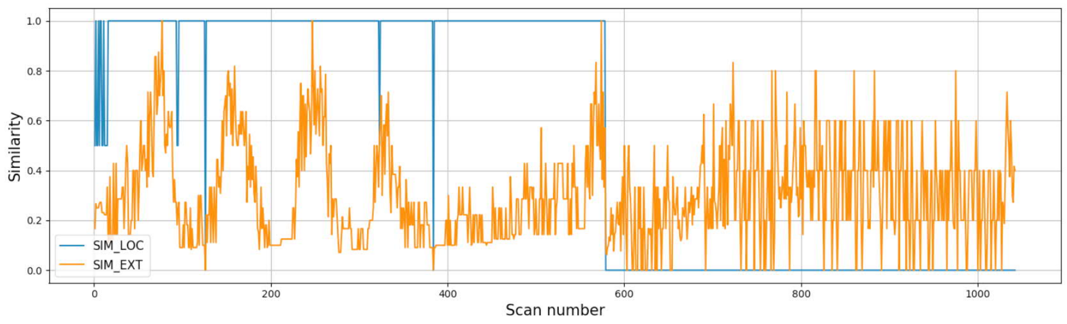

**Figure 7.8.** Scenario 2. $TS_{STA}$ observation by Jaccard Index metric (ownership changed during the 580th scan).

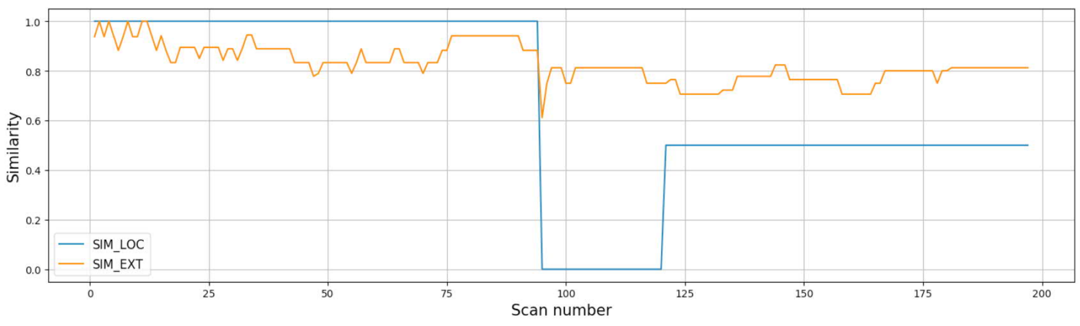

**Figure 7.9.** Scenario 2. $TS_{AP}$ observation by Jaccard Index metric (ownership changed during the 96th scan).

Figures 7.11, 7.12 and 7.13 present the observation of Jaccard Index metric for Scenario 3.

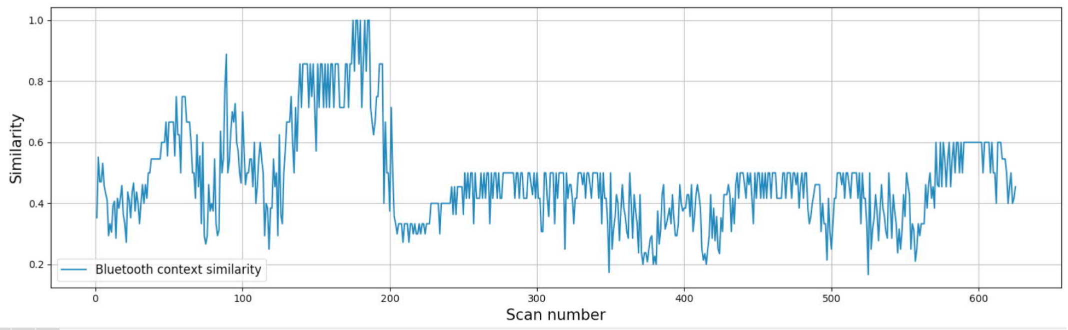

**Figure 7.10.** Scenario 2. $TS_{Bluetooth}$ observation by Jaccard Index metric (ownership changed during the 370th scan).

Jaccard Index metric shows some changes in local context in Figure 7.11, as a new owner connects new devices to the local network during the 460th scan. Meanwhile, there are no significant changes in external context, as the location of the IoT device did not change.

Figure 7.11 depicts Jaccard Index metric observations for $TS_{STA}$ for Scenario 3. As we can see, the metric did not observe a significant drop in context similarity, as there were still previously detected devices in the network.

*FoundIoT* supports Scenario 3 by implementing the multi-stage detection scheme. In case ownership change is not detected by analyzing $TS_{STA}$, the system will try to detect contextual changes in $TS_{AP}$. Figure 7.12 shows Jaccard Index observations for $TS_{AP}$. The metric shows significant drop during the 98th scan after a new owner connects to a new WiFi access point.

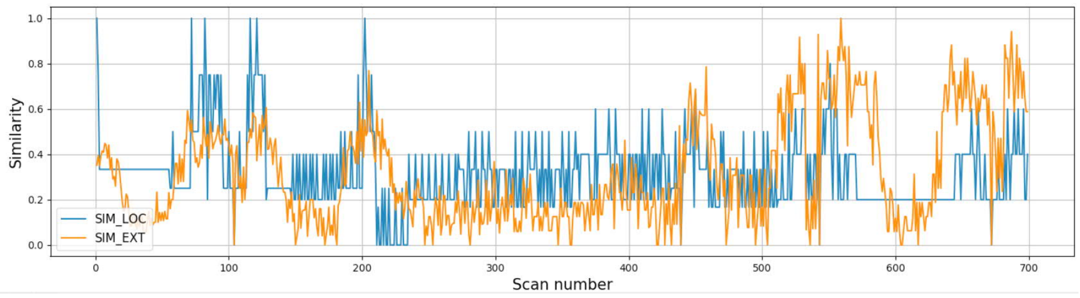

**Figure 7.11.** Scenario 3. $TS_{STA}$ observation by Jaccard Index metric (ownership changed during the 460th scan).

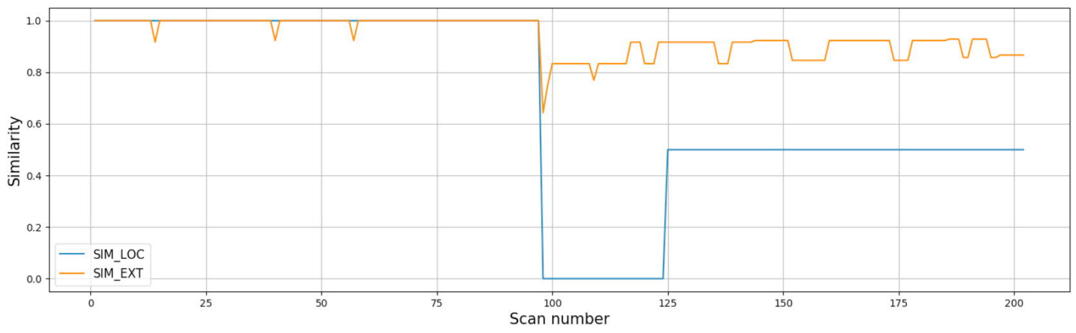

**Figure 7.12.** Scenario 3. $TS_{AP}$ observation by Jaccard Index metric (ownership changed during the 98th scan).

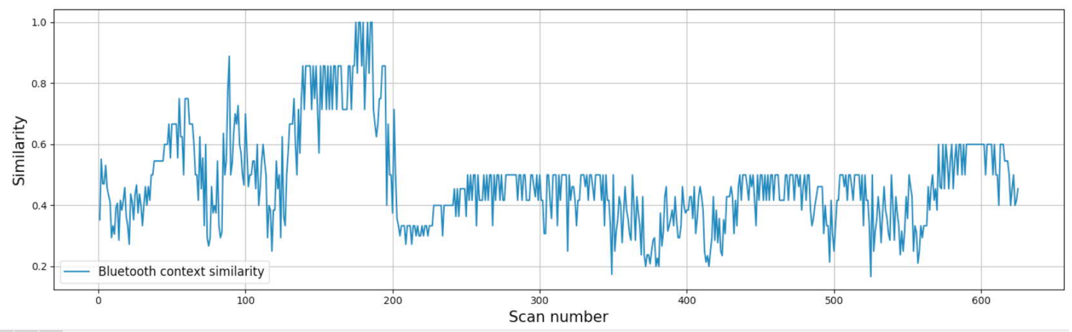

**Figure 7.13.** Scenario 3. $TS_{Bluetooth}$ observation by Jaccard Index metric (ownership changed during the 370th scan).

### 7.3.2 Accuracy Evaluation

As it was previously stated in Section 5.3, values of detection threshold $\Delta_{similarity}$ and window $W$ affect the system accuracy. We evaluate accuracy in terms of TPR and FPR. According to requirement **R3.1**, the system needs to detect ownership change with at least 95% accuracy and false positive rate lower than 1%.

We produce experimental data from two datasets of collected contexts that represent two distinct environments described in Ownership Change Scenarios. Each dataset consists of 7 days of scans. We mix the days to achieve several consecutive ownership changes. The resulted experimental dataset is depicted in Figure 7.14. Each pair of days represent ownership change. As a result, the system needs to detect 13 consecutive ownership changes to achieve the highest accuracy.

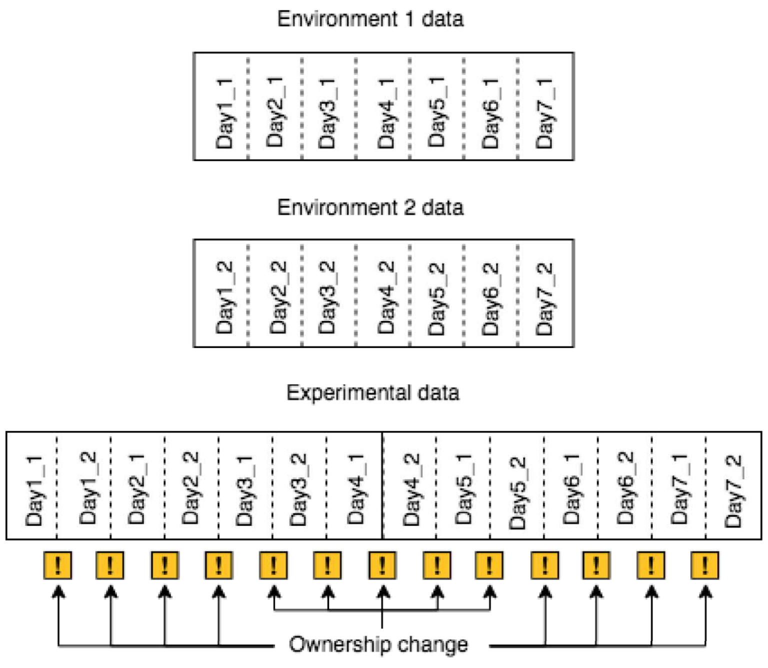

**Figure 7.14.** Experimental Data Generation

For example, Figure 7.15 presents ***FoundIoT*** accuracy evaluation for $TS_{STA}$

in case of Scenario 1 using Jaccard Index metric. Blue labels specify the used detection threshold for local context similarity and window, red labels - detection threshold for external context similarity and window. As we can see, each combination of detection threshold and window has a unique impact on system accuracy. The system achieves the highest accuracy with TPR of 100% and FPR of 0% using the window $W$ of 12 scans and 0.7 and 0.3 as the values of detection thresholds for local and external context similarity, respectively.

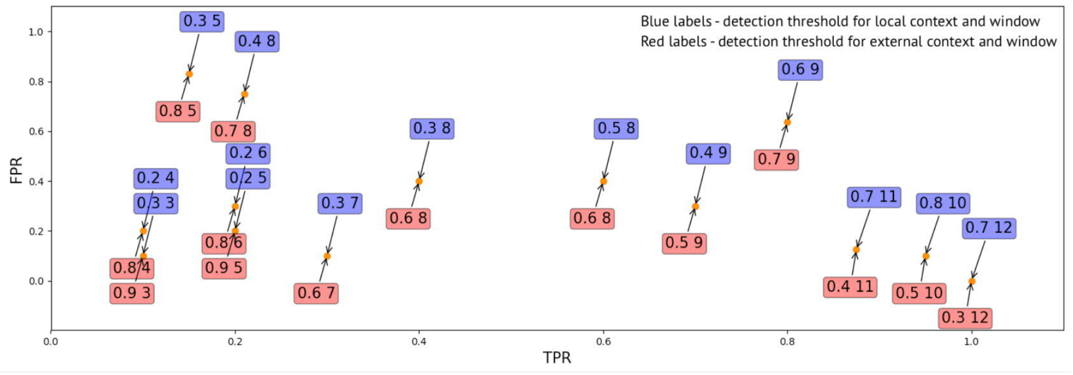

**Figure 7.15.** Scenario 1. Accuracy Evaluation for $TS_{STA}$ using Jaccard Index in terms of TPR and FPR

Table 7.5 presents the highest accuracy values of *FoundIoT* detection using Jaccard Index achieved in case of each ownership change scenario.

| **Analyzed** $TS$ | $1^{st}$ **scenario** | $2^{nd}$ **scenario** | $3^{rd}$ **scenario** |
|---|---|---|---|
| $TS_{STA_{loc}}$ | TPR:100%, FPR:0% 12 scans (1h43m) | TPR:100%, FPR:0% 12 scans (1h43m) | TPR:72%, FPR:22% 12 scans (1h43m) |
| $TS_{STA_{ext}}$ | TPR:100%, FPR:0% 12 scans (1h43m) | TPR:82%, FPR:14% 12 scans (1h43m) | TPR:73%, FPR:23% 12 scans (1h43m) |
| $TS_{AP_{loc}}$ | TPR:100%, FPR:0% 9 scans (1h6m) | TPR:100%, FPR:0% 9 scans (1h6m) | TPR:100%, FPR:0% 9 scans (1h6m) |
| $TS_{AP_{ext}}$ | TPR:100%, FPR:0% 9 scans (1h6m) | TPR:78%, FPR:1% 9 scans (1h6m) | TPR:72%, FPR:2% 9 scans (1h6m) |
| $TS_{Bluetooth}$ | TPR:100%, FPR:0% 10 scans (1h24m) | TPR:79%, FPR:14% 10 scans (1h24m) | TPR:70%, FPR:20% 10 scans (1h24m) |

**Table 7.5.** Performance Evaluation of *FoundIoT* Detection Using Jaccard Index Metric for Analyzed $TS$

Jaccard Index metric shows the highest accuracy with TPR of 100% and FPR of 0% for each $TS$ with similarity scores for each analyzed context in case of Scenario 1. To achieve the highest accuracy for each $TS$, Jaccard Index metric requires 12 scans, which takes 1 hour 43 minutes. Jaccard Index metric is able to detect ownership change very accurately because of completely different environments that represent Scenario 1. As a result, Jaccard Index metric can detect changes in the IoT device WiFi and Bluetooth contexts in Scenario 1 and achieves the accuracy required by **R3.1**.

In case of Scenario 2, Jaccard Index metric provides the highest accuracy with

TPR of 100% and FPR of 0% only for $TS_{STA_{loc}}$ and $TS_{AP_{loc}}$ because of different local WiFi networks of the previous and new owners and no overlap in the devices that are connected to them. Jaccard Index metric requires a window of 12 scans to achieve the required accuracy, which takes 1 hour 43 minutes. In case of $TS_{STA_{ext}}$, $TS_{AP_{ext}}$ and $TS_{Bluetooth}$, Jaccard Index metric achieves lower accuracy with TPR of 78% - 82% and FPR of 1% - 14% because of overlap in the devices that are located in the IoT device vicinity due to close location of the new owner in Scenario 2. As a result, we can rely only on Jaccard Index metric values for local WiFi ($TS_{STA_{loc}}$ and $TS_{AP_{loc}}$) context to achieve the accuracy required by **R3.1**.

Finally, in case of Scenario 3, Jaccard Index metric achieves the highest accuracy with TPR of 100% and FPR of 0% only for $TS_{AP_{loc}}$ because of a new AP in the local network. The requried accuracy is achieved by using a window of 9 scans, which takes 1 hour 6 minutes. Lower accuracy values with TPR of 70% - 73% and FPR of 2% - 23% for other time series are caused by a large overlap in devices from the previous owner and new owner contexts due to unchanged location of the IoT device. As a result, Jaccard Index metric provides the accuracy required by **R3.1** only for $TS_{AP_{loc}}$.

To conclude Jaccard Index metric accuracy evaluation, we can state that it achieves the accuracy required by **R3.1** only for $TS_{AP_{loc}}$ in each scenario. Analysis of other time series provides lower accuracy due to an overlap in the previous owner and new owner contexts. As a result, to detect ownership change accurately using Jaccard Index metric, we need to use $TS_{AP_{loc}}$ and disregard other contexts.

Next, we present the evaluation of *FoundIoT* detection accuracy using KL Divergence. Table 7.6 presents achieved accuracy values in case of each ownership change scenario.

| **Analyzed** $TS$ | $1^{st}$ **scenario** | $2^{nd}$ **scenario** | $3^{rd}$ **scenario** |
|---|---|---|---|
| $TS_{STA_{loc}}$ | TPR:100%, FPR:0% 9 scans (1h6m) | TPR:100%, FPR:0% 9 scans (1h6m) | TPR:74%, FPR:19% 9 scans (1h6m) |
| $TS_{STA_{ext}}$ | TPR:100%, FPR:0% 9 scans (1h6m) | TPR:83%, FPR:12% 9 scans (1h6m) | TPR:75%, FPR:21% 9 scans (1h6m) |
| $TS_{AP_{loc}}$ | TPR:100%, FPR:0% 7 scans (48m) | TPR:100%, FPR:0% 7 scans (48m) | TPR:100%, FPR:0% 7 scans (48m) |
| $TS_{AP_{ext}}$ | TPR:100%, FPR:0% 7 scans (48m) | TPR:76%, FPR:1% 7 scans (48m) | TPR:75%, FPR:2% 7 scans (48m) |
| $TS_{Bluetooth}$ | TPR:100%, FPR:0% 8 scans (57m) | TPR:75%, FPR:11% 8 scans (57m) | TPR:73%, FPR:18% 8 scans (57m) |

**Table 7.6.** Performance Evaluation of *FoundIoT* Detection Using KL Divergence Metric for Analyzed $TS$

As we can see from Table 7.6, KL Divergence metric achieves similar accuracy values to Jaccard Index. It achieves the accuracy required by **R3.1** only for $TS_{AP_{loc}}$ in each scenario. For other contexts, KL Divergence shows lower accuracy and does not reach required accuracy. The main advantage of KL Divergence is that it requires a lower window of scans to achieve the required accuracy compared to Jaccard Index, which leads to the increase in detection speed by 25-30 minutes on average for each analyzed context.

To conclude the accuracy evaluation, we can state that the system meets the requirement **R3.1** for each ownership change scenario using Jaccard Index and KL Divergence metrics when analyzing $TS_{AP_{loc}}$, which represents the similarity of local APs in the IoT device vicinity. Both metrics show worse accuracy when analyzing other contexts due to overlap in discovered devices and cannot be used for ownership change detection as they do not meet requirement **R3.1**.

### 7.3.3 Speed Evaluation

The speed of ownership change detection depends on the parameter $W$, which sets the window of consecutive scans with similarity metric lower than $\Delta_{similarity}$. Delays in detection speed lead to the increase of the window for an attack to occur.

The current implementation of *FoundIoT* sets a delay of 10 minutes between the scans. The final value of detection speed includes this delay. Different values of window $W$ cause a unique impact on the accuracy of detection. For example, Figure 7.16 presents the speed comparison chart with detection speed using Jaccard Index and KL Divergence metrics for $TS_{STA}$ in Scenario 1. KL Divergence shows higher detection speed by 37 minutes compared to Jaccard Index.

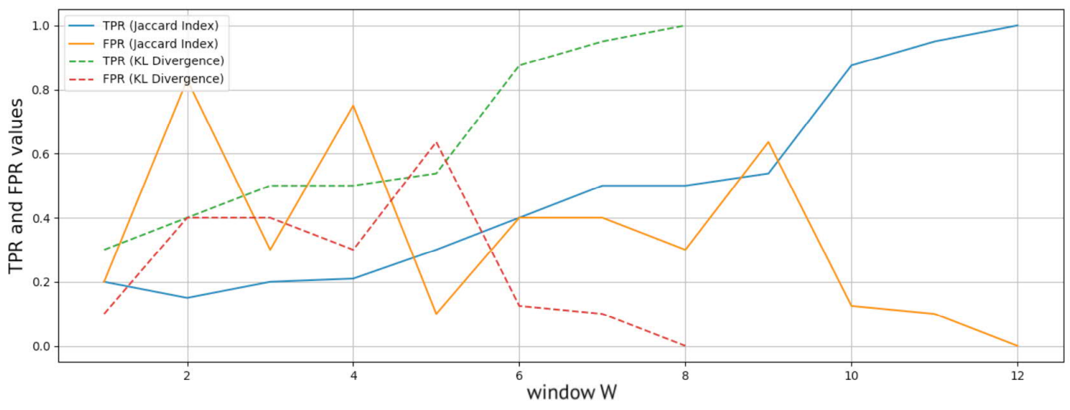

**Figure 7.16.** Scenario 1. Detection speed comparison of Jaccard Index and KL Divergence metrics for $TS_{STA}$. The speed of detection is 1 hour 43 minutes and 1 hour 6 minutes for Jaccard Index and KL Divergence metrics, respectively.

Tables 7.5 and 7.6 present the highest speed values achieved in each ownership change scenario. In case of Scenario 1, Jaccard Index and KL Divergence metrics require 12 (1 hour 43 minutes) and 9 (1 hour 6 minutes) consecutive scans to achieve TPR of 100% and FPR of 0% for $TS_{STA}$ analysis, respectively. In case $TS_{AP}$ analysis, Jaccard Index metric and KL Divergence require 9 (1 hour 6 minutes) and 7 (48 minutes) scans, respectively. And finally, in case of $TS_{Bluetooth}$ analysis, Jaccard Index metric and KL Divergence require 10 (1 hour 24 minutes) and 8 (48 minutes) scans, respectively. As a result, KL Divergence provides higher detection speed than Jaccard Index, which allows the system to detect ownership change faster.

For Scenario 2, Jaccard Index and KL Divergence metrics provide the same speed values as for Scenario 1. Jaccard Index and KL Divergence metrics require 12 (1 hour 43 minutes) and 9 (1 hour 6 minutes) consecutive scans for $TS_{STA}$, 9 (1 hour 6 minutes) and 7 (48 minutes) scans for $TS_{AP}$, and 10 (1 hour 24 minutes) and 8 (57 minutes) scans for $TS_{Bluetooth}$, respectively. The speed values that are provided by KL Divergence metric are consistently higher compared to Jaccard Index.

Finally, for Scenario 3, Jaccard Index and KL Divergence metrics provide consistent speed values as for the previously evaluated scenarios. KL Divergence metric provides higher speed compared to Jaccard Index for each analyzed context. However, the achieved detection speed does not meet requirement **R3.2**.

As a result, we can state that KL Divergence provides higher detection speed compared to Jaccard Index by 25-30 minutes on average for all analyzed contexts in each scenario. However, the current implementation does not meet requirement **R3.2** that was set in Section 3.3.

## 7.4 Re-evaluation using AP Detection technique

Sections 7.3.2 and 7.3.3 presented the accuracy and speed evaluation of the current implementation of *FoundIoT*. According to the gathered results, *FoundIoT* achieves the highest accuracy with TPR of 100% and FPR of 0% and highest speed of 48 minutes using KL Divergence metric. However, it does not meet the speed requirement set in Section 3.3 due to delay of 10 minutes between the scans and the multi-stage ownership change detection scheme.

According to accuracy evaluation presented in Section 7.3.2, $TS_{AP_{loc}}$ provides the highest accuracy in each examined scenario. That is why, we have limited the ownership change detection scheme with AP Detection technique. In this section, we re-evaluate the system performance by making new experiments.

### 7.4.1 Experiment setup

The setup is composed of our IoT testbed device connected to a WiFi AP, which is called local, and external APs. The testbed device invokes iwlist utility to scan for APs in its vicinity periodically, each 30 seconds. Each scan takes on average 3 seconds. As a result, each scan generates two time series: local and external APs. The scans are performed in different locations to check the system performance for all ownership change scenarios described in Section 7.3.1.

We also change the number of scans used for generation of set of previously discovered devices $Set_{prev}$. $Set_{prev}$ is used in the computation of the similarity score by Jaccard Index and KL Divergence metrics. In the initial implementation evaluated in Sections 7.3.2 and 7.3.3, we generate $Set_{prev}$ based on all prior scans. In our new experiment setup, we use only 10 previous scans to generate $Set_{prev}$.

### 7.4.2 Accuracy Evaluation

In this section, we evaluate the system detection accuracy.

As it was mentioned earlier in Section 5.3, the accuracy of detection depends on two parameters: detection threshold $\Delta_{similarity}$ and window $W$. To evaluate the accuracy of KL Divergence metric, we use the approach described earlier in Section 7.3.2 to generate experimental dataset with mixed data from different contexts.

For example, Figures 7.17 and 7.18 present the accuracy and speed evaluation of *FoundIoT* detection for Scenario 3. The system reaches the highest accuracy with TPR of 70% and FPR of 1% for $TS_{AP_{ext}}$ in 3 scans, which takes 1 minute 39 seconds. KL Divergence cannot identify changes in the IoT device $TS_{AP_{ext}}$ context accurately due to unchanged location of the IoT device, which leads to low TPR values.

The system meets requirement **R3.2**; however, it does not meet requirement **R3.1** due to low TPR values. As a result, this means that KL Divergence for $TS_{AP_{ext}}$ can be used as a supplementary technique for detection of ownership change.

Table 7.7 presents the highest accuracy values of *FoundIoT* detection using KL Divergence achieved in case of each ownership change scenario in our new experiment.

| **Analyzed** $TS$ | $1^{st}$ **scenario** | $2^{nd}$ **scenario** | $3^{rd}$ **scenario** |
|---|---|---|---|
| $TS_{AP_{loc}}$ | TPR:100%, FPR:0% 1 scan (33sec) | TPR:100%, FPR:0% 1 scan (33sec) | TPR:100%, FPR:0% 1 scan (33sec) |
| $TS_{AP_{ext}}$ | TPR:100%, FPR:0% 1 scan (33sec) | TPR:100%, FPR:0% 2 scans (1min6sec) | TPR:70%, FPR:1% 3 scans (1min39sec) |

**Table 7.7.** Performance Evaluation of *FoundIoT* Detection Using KL Divergence Metric for $TS_{AP_{loc}}$ and $TS_{AP_{ext}}$

KL Divergence metric reaches the accuracy required by **R3.1** in all ownership change scenarios. However, due to unchanged location of the IoT device and

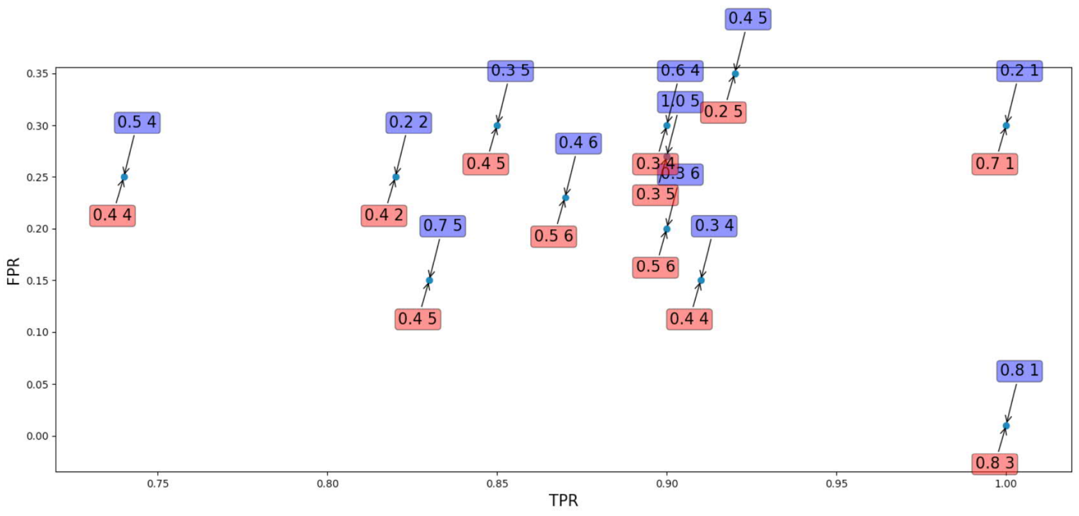

**Figure 7.17.** Scenario 3. Detection Threshold and Window Parameters Evaluation for $TS_{AP_{ext}}$ using KL Divergence metric.

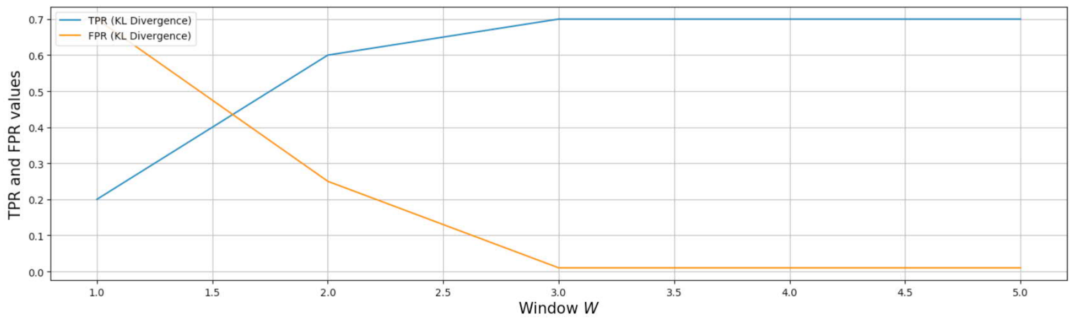

**Figure 7.18.** Scenario 3. Detection speed evaluation using KL Divergence metric for $TS_{AP_{ext}}$.

large overlap of devices that belong to the previous and new owner contexts, KL Divergence cannot accurately identify difference in the external contexts $TS_{AP_{ext}}$ in Scenario 3 and reaches only TPR of 70% and FPR of 1%. For $TS_{AP_{loc}}$, KL Divergence provides the required accuracy because of different APs used by the previous owner and a new owner. As a result, we can use external and local $TS_{AP}$ for ownership change detection because both of them does not cause FPR higher than 1% as required by **R3.1**. In case the system does not detect ownership change in the external context, it switches to the local context, which provides the required TPR.

It is worth mentioning that in our new experiment, KL Divergence is able to achieve the required accuracy with a significantly higher speed. This can be explained by the reduced number of scans used for generation of $Set_{prev}$. Thus, KL Divergence performs better on sets with the lower number of devices and achieves the required accuracy under 2 minutes.

As a result, we can state KL Divergence metric provides the accuracy required by **R3.1** in each ownership change scenario.

### 7.4.3 Speed evaluation

The evaluation of accuracy presented in Section 7.4.2 showed that KL Divergence metric can detect ownership change with highest accuracy using the window of 3 scans. Due to delay of 30 seconds between the scans, the lowest speed of *FoundIoT* detection is 1 minute 39 seconds, which satisfies our speed requirement **R3.2**.

As a result, we can state the described technique fully meets the requirement **R3.2**.

## 7.5 Security

### 7.5.1 MAC Address Spoofing

One of security issues of the current approach is related to MAC address spoofing attack. *FoundIoT* detects and identifies wireless context in the device vicinity during capturing phase. The identification is based on MAC address of device network card. This information could be easily spoofed by an adversary.

One possible solution for MAC address spoofing is device fingerprinting. J. Franklin et al. [36] described a passive device driver fingerprinting technique that is based on different network card 802.11 implementations by vendors. With the use of developed technique, authors were able to reliably identify 802.11 chipsets and even specific version of network card driver. The authors' general idea of the fingerprinting relies on different IEEE 802.11 [6] implementations by device manufacturers, which could lead to the identification of a device driver, firmware or even user applications that use the network channel. 802.11 implementation implies an active scanning procedure, which is used by a device to scan its vicinity for an AP. A device sends a specific management frame called the probe request. If an AP is compatible with the scan parameters of a device, it sends a probe response to acknowledge received request. Due to different 802.11 implementations, devices from different vendors send management frames with varying frequency. This feature helps to differentiate the device vendor. One of the most important features of 802.11 fingerprinting is that the implementation is mainly controlled by hardware. It means that in most cases it is impossible to mimic a network card 802.11 implementation and spoof the context by an adversary. However, the fingerprinting approach cannot be used for our system, as it requires scans with a long duration to collect statistical data for accurate fingerprint generation. This prevents the system from respecting the speed requirement **R3.2**.

Sheng et al. [37] and Chen et al. [38] proposed solutions that analyze the signal strength of the monitored station. The algorithm detects a spoofed station based on the difference of signal strength of reply frames sent back from the genuine and spoofed stations. This approach is feasible for the IoT device ownership change scenario as it does not require a lot of scans to identify the signal strength of nearby wireless stations.

Bharti et al. [39] proposes a security association approach for MAC address spoofing prevention. The idea implies sharing a secret key that is associated with a specific AP. During data transmission, the AP sends a packet that is encrypted using the secret key to the node. In case the node cannot decrypt the packet, the AP is assumed to be fake. This approach is more secure than the previously described techniques as an adversary does not know the pre-shared secret and cannot spoof the AP. However, it requires more computational power because it involves encryption. This approach can be used for the IoT device ownership change scenario to ensure the authenticity of the local AP and mitigate MAC address spoofing attacks.

### 7.5.2 Ownership Change Detection Delay

*FoundIoT* involves AP Detection technique to detect ownership change. As it was shown in Section 7.4.3, *FoundIoT* is able to detect ownership change with the use of AP Detection technique with the lowest speed of 1 min 39 seconds.

This technique detects change of the IoT device ownership during its initial configuration, and therefore does not let the device and sensitive information stored on it become vulnerable to attacks from an adversary. Mitigation of adversary actions is very important, as he has full control over the device during ownership change. An adversary can try to extract previous owner's sensitive information stored on the device, which includes account and network credentials, and information collected by the IoT device during its operation once it is configured.

### 7.5.3 Susceptibility to Power Cuts

Since most smart home devices do not have their own power source, they are susceptible to power cuts. In such cases, devices stop operating and turn off completely. Upon power restoration, the device re-initializes itself. Re-initialization includes initiating detection of ownership change, restoring Internet connection, and connecting to the control device.

In case of power cut during operation of *FoundIoT*, the execution will be continued as soon as power is restored. If the monitor component completes its scan, the collected contextual information will be saved and used for analysis during next iteration. If the detection condition is respected during the scan after power is restored, the system will transfer the control to chownIoT for secure data handling.

While the IoT device is turned off, an adversary can try to launch a physical attack to manipulate the collected context saved on the IoT device storage and disrupt the detection technique. Currently, there is no direct protection against physical attacks. One possible solution to mitigate this attack is computation of checksums for saved contexts. In case the system detects the checksum mismatch, it initiates the secure data handling by chownIoT.

# 8. Related work

Recently, several researchers have focused on ensuring sensitive information privacy and providing security improvements for IoT devices. Our proposed solution is primarily focused on detecting change of IoT device ownership by registering context changes to further ensure the privacy of sensitive data stored on smart home devices.

We organize Chapter 8 in the following way: Section 8.1 discusses alternative solutions that provide smart home device privacy. In Section 8.2, we discuss research works related to context aware security.

## 8.1 Smart Home Device Privacy

The prevalence of smart home devices is increasing exponentially, which allows to execute daily routines more conveniently. However, it is also introducing new unique challenges for preserving the owner's privacy in the smart home environment. Therefore, recent studies have been focusing on ensuring privacy of smart home device owners. Several of these works are described below.

Song et al. [40] proposed a solution that helps to preserve privacy during communication between smart home appliances and the control device. The authors mainly focused on eavesdropping prevention from an outside adversary by proposing a secure communication protocol for smart home devices. In the proposed protocol, data is encrypted before the transfer to achieve confidentiality. To ensure data integrity, the protocol appends Message Authentication Code (MAC) to messages for detection of tampering and data modification by an adversary. Finally, the protocol uses MAC for two-way authentication. Smart home devices ensure that a received message comes from a certain control device and a control device can prove that a message comes from a legitimate smart home device. As a result, authors described an architecture for a smart home system to ensure user's data security and privacy. However, the proposed solution does not cover situations of ownership change. If the smart home device is sold to a different owner or stolen, the previous owner's sensitive information will be accessible to a new owner or an adversary.

Sivaraman et al. [41], Liu et al. [42] and Yoshigoe et al. [43] studied the operation of popular smart home appliances, identified several security and privacy issues, and proposed an approach for improving security system of the smart home device and mitigating privacy threats. The solution operates on the network level and blocks any suspicious activities for the current environment. It provides a capability to the owner to specify a set of permitted actions for a certain environment. The system can either grant or prohibit an action, which the device tries to execute, based on the current environment. However, the proposed solutions do not take into account ownership change situations, which introduce unique challenges to preserve data privacy.

Lee et al. [44] presented a different approach for preserving privacy for smart home devices based on a cloud platform. The proposed solution describes a special node, *central home controller*, that provides data-hiding and data analytical access control capabilities in the cloud. The system enables easy access and understanding for users and manufacturers that collect diagnostic data about the device operation for service improvement. Therefore, it ensures device owner's privacy and data availability for manufacturers. However, authors do not examine issues related to change in the smart home device ownership. The proposed solution does not provide mechanisms for preserving privacy in case the owner of the device has changed or the device was stolen. Home central controller will reveal information stored in its memory or the cloud storage to an adversary or a new owner.

The described works are ensuring privacy of smart home device owner without considering privacy issues during ownership change. The solutions that focus on preserving data privacy for IoT devices need to act differently based on their context as most IoT smart home devices are portable and can be easily taken to a different location by an adversary. On the contrary, our solution is focused on addressing privacy issues related to ownership change of smart home devices to further ensure privacy of data stored on the IoT device.

## 8.2 Context Aware Security

During the last 10 years, the number of mobile devices and their adoption rate have been constantly increasing. These devices include smartphones, tablets, smart watches and smart home IoT devices. These devices are equipped with a variety of different sensor modalities that can detect their context.

The contextual information can be used for various tasks and provide context aware services. Some services provide security solutions that use contextual information to improve security of IoT devices. Some of the recent research works focused on context aware security are presented below.

The idea of providing context based access control has been a rather popular area of research in recent years. Several works have described techniques for applying context to provide access control. Wullems et al. [45] proposed an authorization architecture for providing access control to resources based on host context, such as its location, network topology, host security (OS patches, firewall rules, filesystem permissions). Chung et al. [46] suggested a context-aware security model based on user and network environment changes. In [47], Hu et al. developed a context-aware framework for health care systems that is aimed to provide access control to hospital resources based on user context and trust level assigned by the hospital administrator. Shebaro et al. [48] and Das et al. [49] introduced similar solutions that provide access control based on user's location and his activities. The proposed systems monitor access requests from applications on a mobile device and grants or deny access using rule policy for the current context. In [50], Cantali et al. presented a context-aware security system for providing Internet access. The authors include properties, such as wireless security protocols, encryption algorithms, user's location and authentication protocols, for context identification. As a result, the security system rates wireless networks in uncontrolled public spaces with a low assessment criteria, which does not let the mobile device to connect to them. Satoh [51] introduced a model for context-aware access control executed on the cloud computing side. The model treats user's location as the primary factor for providing access to control smart home appliances only for residents that are located in the same room with them. Gupta et al. [52] proposed a context-based framework for access control. The proposed solution collects GPS, Bluetooth and WiFi data to detect current context and computes its familiarity and safety scores. The context-based framework has been integrated with a device locking application. The software is able to accurately detect contextual changes and adjust the safety level accordingly. Covington et al. [53] described an advanced model for securing context-aware environments and providing flexible access control. The model supports context-aware authorization, which means that access could be restricted based on contextual factors, such as subject's location, room temperature or the time of day. Another feature of the model is related to non-intrusive user authentication based on voice and face recognition.

The concept of non-intrusive user authentication was also discussed by Truong et al. [20] when developing a context-aware zero-interaction authentication (ZIA).

ZIA refers to an approach that supports user authentication without any interaction with a verifier (terminal). The authors used different sensor modalities (WiFi, Bluetooth, GPS and audio) for device and verifier co-presence detection and experimentally proved that WiFi is the module that provides the most effective resistance to relay attacks. The major contributions of the paper are the context detection evaluation of different sensor modalities and the use of multiple modalities to improve resilience against relay attacks without degrading usability.

Ashibani et al. [54] introduced a context-aware authentication framework for smart home environment. The proposed solution treats contextual information, such as user's location, profile and request time to grant or deny access. The framework provides user authentication in a non-intrusive manner, it bases completely on contextual information without requiring the user to provide credentials.

Previously mentioned works focused on authentication and authorization tasks to either control access to a resource or ensure authenticity of the user. In contrast, Al-Rabiaah et al. [55] described a mechanism that validates the device sensor (context collector) and ensures integrity of collected context information.

Besides access control and user authentication, in another work, Harb et al. [56] proposed a solution for secure group key management based on contextual information. The main motivation for the proposed solution is IoT devices resource constraints. Due to limited resources, authors suggest to rely on multicast communication to transmit data to several receivers. To secure the transmission, the data needs to be encrypted. The proposed solution shares the group key used by sender and the receivers based on collected context on each communication member. The contextual information includes wireless network details (WiFi, 3G, Zigbee), device geographical location and the multicast group to join.

All previously mentioned works use context to provide access control, user authentication, authorization or to establish secure communication and eventually improve device security. In our work, we use contextual information to detect change of the IoT device ownership. None of the works prior ours have used context of a device to detect change of its ownership. The use of contextual information to detect ownership change provides the capability to a smart home device to take necessary security measures and ensure data privacy automatically.

# 9. Conclusions

## 9.1 Summary of Contributions

***Automatic ownership change detection system:*** We have presented a system called *FoundIoT* that detects ownership change of the IoT device based on contextual changes. The proposed solution uses the device WiFi sensor modality for context identification. It associates changes of the collected device context with change of its ownership.

***Data mining techniques:*** We have developed and presented data mining techniques based on Jaccard Index and KL Divergence metrics that are applied to the collected wireless communication data. *FoundIoT* relies on these techniques for ownership change detection.

***A prototype implementation:*** We have presented a prototype implementation of the proposed system. We have used the Raspberry Pi board for deploying the system and evaluating its performance. The system operates completely autonomously on the device and does not require any additional external dependencies.

The implementation also helps identifying practical requirements and overall system performance.

***Evaluation:*** We have provided the system performance evaluation, by measuring speed and accuracy of the IoT device ownership change detection. The implemented prototype is able to detect ownership change with high accuracy and low number of false alarms. The system can achieve TPR of 100% and FPR of 1% with speed up to 1 minute 39 seconds. High speed of ownership change detection helps the system to mitigate adversary attacks to some extent. However, the current implementation is vulnerable to MAC address spoofing attack. This problem has been discussed by several works and lightweight solutions were proposed to mitigate this attack.

## 9.2 Future work

***Protection against spoofing:*** The current implementation of *FoundIoT* relies on context identification using WiFi communication channel. The system captures MAC addresses of active APs in the vicinity of the IoT device. As it was discussed in Section 7.5.1, these readings can be spoofed by an adversary to manipulate the context. Therefore, mitigation of this security fault is one of the main future development directions for the project.

One of the techniques that should not create system overload and significantly decrease the speed of ownership change detection is device fingerprinting. This technique is based on the device specific features that cannot be spoofed.

***Developing a communication protocol with chownIoT:*** Currently, *FoundIoT* operates independently and provides a response of whether ownership change was detected or not. To ensure sensitive data privacy on the IoT device during ownership change, we need to develop a communication protocol for *FoundIoT* and chownIoT.

The protocol needs to provide a capability to *FoundIoT* to communicate with chownIoT when ownership change of the IoT device is detected. As a result, chownIoT needs to activate ownership change management, which includes ensuring privacy of the previous owner's sensitive information with encryption, preparing the device for using by a new owner and restoring the context when the device is returned to the previous owner.

***The complete system deployment on the IoT testbed device:*** Currently, both components of the complete system, *FoundIoT* and chownIoT, have a prototype implementation on the testbed IoT device based on the Raspberry Pi board. Both components have been individually evaluated in terms of their system load and performance. It has been shown that each component operates effectively. However, we need to ensure the complete system performance.

# Bibliography

[1] Thien Duc Nguyen et al. “DÏoT: A Crowdsourced Self-learning Approach for Detecting Compromised IoT Devices”. In: *CoRR* abs/1804.07474 (2018). arXiv: 1804.07474. URL: http://arxiv.org/abs/1804.07474.

[2] The Bluetooth SIG. *BluetoothLE*. Version 4.0. 2010. URL: https://www.bluetooth.com.

[3] Zensys. *Z-Wave*. 2001. URL: http://www.z-wave.com.

[4] Open Connectivity Foundation. *IoTivity*. Version 1.3.1. Dec. 18, 2017. URL: https://www.iotivity.org.

[5] Apple Inc. *HomeKit*. Sept. 2014. URL: https://developer.apple.com/homekit/.

[6] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*. Piscataway, NJ, 2012.

[7] The Tcpdump team. *Tcpdump*. Version 4.9.2. Sept. 3, 2017. URL: https://tcpdump.org.

[8] Gerald Combs, The Wireshark team. *WireShark*. Version 2.4.5. Feb. 24, 2018. URL: https://wireshark.org.

[9] Markus Miettinen et al. “ConXsense: automated context classification for context-aware access control”. In: *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM. 2014, pp. 293–304.

[10] Md Khan et al. “Enhancing Privacy in IoT Devices through Automated Handling of Ownership Change”. In: (2017).

[11] *Raspberry*. https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/. Accessed: 2018-05-08.

[12] Brian Fox. *Bash*. Version 4.4.18. Jan. 30, 2018. URL: www.gnu.org/software/bash/.

[13] Python Software Foundation. *Python*. Version 3.6.5. Mar. 28, 2018. URL: https://www.python.org.

[14] Andrei Costin, Apostolis Zarras, and Aurélien Francillon. “Automated dynamic firmware analysis at scale: a case study on embedded web interfaces”. In: *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM. 2016, pp. 437–448.

[15] Daniel Fraunholz et al. “Investigation of cyber crime conducted by abusing weak or default passwords with a medium interaction honeypot”. In: *Cyber Security And Protection Of Digital Services (Cyber Security), 2017 International Conference on*. IEEE. 2017, pp. 1–7.

[16] *Tessel 2*. https://tessel.io/blog/113259439202/tessel-2-hardware-overview. Accessed: 2018-05-08.

[17] Markus Miettinen et al. “IoT Sentinel: Automated device-type identification for security enforcement in IoT”. In: *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE. 2017, pp. 2177–2184.

[18] Serkan Polat. *Bluetoothctl*. Version 1.1.0. July 31, 2017. URL: https://bitbucket.org/serkanp/bluetoothctl.git.

[19] Paul Jaccard. “Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines”. In: *Bull Soc Vaudoise Sci Nat* 37 (1901), pp. 241–272.

[20] Hien Thi Thu Truong et al. “Using contextual co-presence to strengthen zero-interaction authentication: Design, integration and usability”. In: *Pervasive and Mobile Computing* 16 (2015), pp. 187–204.

[21] Olivier Dousse, Julien Eberle, and Matthias Mertens. “Place learning via direct WiFi fingerprint clustering”. In: *Mobile Data Management (MDM), 2012 IEEE 13th International Conference on*. IEEE. 2012, pp. 282–287.

[22] Solomon Kullback and Richard A Leibler. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[23] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[24] Taiyeong Lee et al. “Clustering Time Series Based on Forecast Distributions Using Kullback-Leibler Divergence”. In: Web. 2014.

[25] Song Liu et al. “Change-point detection in time-series data by relative density-ratio estimation”. In: *Neural Networks* 43 (2013), pp. 72–83.

[26] *Raspbian Jessie operating system for Raspebrry Pi*. Accessed: 2018-05-27. URL: `https://www.raspberrypi.org/downloads/raspbian/`.

[27] *Reference Manual Pages (3PCAP)*. Accessed: 2018-06-13. URL: `https://www.tcpdump.org/manpages/pcap.3pcap.html`.

[28] Jean Tourrilhes. *Iwlist*. Version 29.0. Sept. 17, 2007. URL: `https://hewlettpackard.github.io/wireless-tools/Tools.html`.

[29] *JSON encoder and decoder*. Accessed: 2018-05-27. URL: `https://docs.python.org/3/library/json.html#module-json`.

[30] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. Accessed: 2018-05-27. 2001–. URL: `http://www.scipy.org/`.

[31] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[32] Next Thing Co. *C.H.I.P. personal single-board computer*. 2016. URL: `https://getchip.com/pages/chip` (visited on 03/15/2018).

[33] *Intel Edison*. `https://ark.intel.com/en/products/84572/Intel-Edison-Compute-Module-IoT`. Accessed: 2018-05-08.

[34] *Udoo Neo*. `https://www.udoo.org/udoo-neo/`. Accessed: 2018-05-08.

[35] *ps - report process status*. `http://man7.org/linux/man-pages/man1/ps.1.html`. Jan. 13, 2018.

[36] Jason Franklin et al. "Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting." In: *USENIX Security Symposium*. Vol. 3. 2006, pp. 16–89.

[37] Yong Sheng et al. "Detecting 802.11 MAC layer spoofing using received signal strength". In: *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*. IEEE. 2008, pp. 1768–1776.

[38] Yingying Chen, Wade Trappe, and Richard P Martin. "Detecting and localizing wireless spoofing attacks". In: *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON'07. 4th Annual IEEE Communications Society Conference on*. IEEE. 2007, pp. 193–202.

[39] Abhishek Kumar Bharti and Manoj Chaudhary. "Prevention of Session Hijacking and Ipspoofing with Sensor Nodes and Cryptographic Approach". In: *International Journal of Computer Applications* 76.9 (2013).

[40] Tianyi Song et al. "A privacy preserving communication protocol for IoT applications in smart homes". In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 1844–1852.

[41] Vijay Sivaraman et al. “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*. IEEE. 2015, pp. 163–167.

[42] Jianqing Liu, Chi Zhang, and Yuguang Fang. “EPIC: A Differential Privacy Framework to Defend Smart Homes Against Internet Traffic Analysis”. In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 1206–1217.

[43] Kenji Yoshigoe et al. “Overcoming invasion of privacy in smart home environment with synthetic packet injection”. In: *TRON Symposium (TRONSHOW), 2015*. IEEE. 2015, pp. 1–7.

[44] Ying-Tsung Lee et al. “Privacy-preserving data analytics in cloud-based smart home with community hierarchy”. In: *IEEE Transactions on Consumer Electronics* 63.2 (2017), pp. 200–207.

[45] Chris Wullems, Mark Looi, and Andrew Clark. “Towards context-aware security: An authorization architecture for intranet environments”. In: *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*. IEEE. 2004, pp. 132–137.

[46] Mokdong Chung et al. “Context-Aware Security Services in DAA Security Model”. In: *Advanced Language Processing and Web Information Technology, 2008. ALPIT'08. International Conference on*. IEEE. 2008, pp. 424–429.

[47] Junzhe Hu and Alfred C Weaver. “A dynamic, context-aware security infrastructure for distributed healthcare applications”. In: *Proceedings of the first workshop on pervasive privacy security, privacy, and trust*. Citeseer. 2004, pp. 1–8.

[48] Bilal Shebaro, Oyindamola Oluwatimi, and Elisa Bertino. “Context-based access control systems for mobile devices”. In: *IEEE Transactions on Dependable and Secure Computing* 12.2 (2015), pp. 150–163.

[49] Prajit Kumar Das, Anupam Joshi, and Tim Finin. “Personalizing context-aware access control on mobile platforms”. In: *Collaboration and Internet Computing (CIC), 2017 IEEE 3rd International Conference on*. IEEE. 2017, pp. 107–116.

[50] Gokcan Cantali et al. “Lightweight context-aware security system for wireless Internet access”. In: *Communications and Network Security (CNS), 2015 IEEE Conference on*. IEEE. 2015, pp. 765–766.

[51] Ichiro Satoh. “Toward Access Control Model for Context-Aware Services Offloaded to Cloud Computing”. In: *Reliable Distributed Systems Workshops (SRDSW), 2016 IEEE 35th Symposium on*. IEEE. 2016, pp. 7–12.

[52] Aditi Gupta, Markus Miettinen, and N Asokan. “Using context-profiling to aid access control decisions in mobile devices”. In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*. IEEE. 2011, pp. 310–312.

[53] Michael J Covington et al. “A context-aware security architecture for emerging applications”. In: *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*. IEEE. 2002, pp. 249–258.

[54] Yosef Ashibani, Dylan Kauling, and Qusay H Mahmoud. “A context-aware authentication framework for smart homes”. In: *Electrical and Computer Engineering (CCECE), 2017 IEEE 30th Canadian Conference on*. IEEE. 2017, pp. 1–5.

[55] Sumayah Al-Rabiaah and Jalal Al-Muhtadi. “Consec: Context-aware security framework for smart spaces”. In: *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE. 2012, pp. 580–584.

[56] Hussein Harb et al. “Multicast security model for Internet of Things based on context awareness”. In: *Computer Engineering Conference (ICENCO), 2017 13th International*. IEEE. 2017, pp. 303–309.

# List of Figures

2.1 chownIoT Flow Diagram [10] . . . . . . . . . . . . . . . . . . . . . . 8

4.1 *FoundIoT* System Overview . . . . . . . . . . . . . . . . . . . . . . 16
4.2 *FoundIoT* Ownership Change Detection Flow Diagram . . . . . . 17
4.3 *FoundIoT* Components . . . . . . . . . . . . . . . . . . . . . . . . . 19

7.1 *FoundIoT* persistent storage usage over 1 week of scans . . . . . 34
7.2 Scenario 1 - Selling device or the device theft (new owner in a different location). . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
7.3 Scenario 2 - Selling IoT device to a neighbor. . . . . . . . . . . . . 37
7.4 Scenario 3 - Selling smart home with integrated IoT device . . . 37
7.5 Scenario 1. $TS_{STA}$ observation by Jaccard Index metric (ownership changed during the 580th scan). . . . . . . . . . . . . . . . . 38
7.6 Scenario 1. $TS_{AP}$ observation by Jaccard Index metric (ownership changed during the 81st scan). . . . . . . . . . . . . . . . . . . . . 38
7.7 Scenario 1. $TS_{Bluetooth}$ observation by Jaccard Index metric (ownership changed during the 470th scan). . . . . . . . . . . . . . . 39
7.8 Scenario 2. $TS_{STA}$ observation by Jaccard Index metric (ownership changed during the 580th scan). . . . . . . . . . . . . . . . . 39
7.9 Scenario 2. $TS_{AP}$ observation by Jaccard Index metric (ownership changed during the 96th scan). . . . . . . . . . . . . . . . . . . . . 39
7.10 Scenario 2. $TS_{Bluetooth}$ observation by Jaccard Index metric (ownership changed during the 370th scan). . . . . . . . . . . . . . . 40
7.11 Scenario 3. $TS_{STA}$ observation by Jaccard Index metric (ownership changed during the 460th scan). . . . . . . . . . . . . . . . . 40
7.12 Scenario 3. $TS_{AP}$ observation by Jaccard Index metric (ownership changed during the 98th scan). . . . . . . . . . . . . . . . . . . . . 40
7.13 Scenario 3. $TS_{Bluetooth}$ observation by Jaccard Index metric (ownership changed during the 370th scan). . . . . . . . . . . . . . . 41

7.14 Experimental Data Generation . . . . . . . . . . . . . . . . . . . . . . 41
7.15 Scenario 1. Accuracy Evaluation for $TS_{STA}$ using Jaccard Index in terms of TPR and FPR . . . . . . . . . . . . . . . . . . . . . . . . 42
7.16 Scenario 1. Detection speed comparison of Jaccard Index and KL Divergence metrics for $TS_{STA}$. The speed of detection is 1 hour 43 minutes and 1 hour 6 minutes for Jaccard Index and KL Divergence metrics, respectively. . . . . . . . . . . . . . . . . . . . . 44
7.17 Scenario 3. Detection Threshold and Window Parameters Evaluation for $TS_{AP_{ext}}$using KL Divergence metric. . . . . . . . . . . . . 47
7.18 Scenario 3. Detection speed evaluation using KL Divergence metric for $TS_{AP_{ext}}$. . . . . . . . . . . . . . . . . . . . . . . . . . 47

# List of Tables

7.1 Specifications of IoT Boards . . . . . . . . . . . . . . . . . . . . . . . 33
7.2 CPU and RAM usage of *FoundIoT* monitor components on the Raspberry Pi 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
7.3 CPU and RAM usage of *FoundIoT* Ownership Change Detection component on the Raspberry Pi 3 . . . . . . . . . . . . . . . . . . 34
7.4 IoT devices identifiers used in ownership change scenarios . . . 35
7.5 Performance Evaluation of *FoundIoT* Detection Using Jaccard Index Metric for Analyzed $TS$ . . . . . . . . . . . . . . . . . . . . 42
7.6 Performance Evaluation of *FoundIoT* Detection Using KL Divergence Metric for Analyzed $TS$ . . . . . . . . . . . . . . . . . . . . 43
7.7 Performance Evaluation of *FoundIoT* Detection Using KL Divergence Metric for $TS_{AP_{loc}}$ and $TS_{AP_{ext}}$ . . . . . . . . . . . . . . . . 46