JAIST Repository
https://dspace.jaist.ac.jp/

Title	スマートホームにおける自動運用管理のためのネットワークトラフィック生成フレームワーク
Author(s)	Pham, Van Cu
Citation	
Issue Date	2020-03-25
Type	Thesis or Dissertation
Text version	ETD
URL	http://hdl.handle.net/10119/16648
Rights	
Description	Supervisor:丹　康雄，先端科学技術研究科，博士

JAPAN ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Network Traffic Generation Framework for Automated Operation, Administration, and Maintenance in Smart Homes

PHAM VAN CU

Japan Advanced Institute of Science and Technology

Doctoral Dissertation

Network Traffic Generation Framework for Automated Operation, Administration, and Maintenance in Smart Homes

PHAM VAN CU

Supervisor : Professor Yasuo TAN

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
Information Science
March, 2020

Copyright © 2020 by PHAM VAN CU

Abstract

The term *Operation, Administration, and Maintenance* (OAM) represents a group of network management activities that include fault detection, isolation, and restoration in order to keep a network operate stability and reliably. Generally, the OAM is essential for Internet Service Provider networks (commercial networks) and is accountable by experienced network operators with the support of tools and network equipment. Since networking is getting more complex due to the development of the information and communication technologies (ICT), automated OAM which relies on the *Artificial Intelligence for IT Operations* (AIOps) is a must to reduce the workload of human. The AIOps automates IT operations by utilizing concepts of big data analytics and machine learning to analyze data in order to detect and react to issues automatically. Smart homes are homes which utilize home appliances and a home network to connect those appliances to organize residential infrastructure so as to improve the overall quality of life of residents and assist them to live actively and independently. Following the development of the Internet of Things, home networks are becoming more and more sophisticated. Unlike, commercial networks which are monitored by experienced network operators, complex home network systems are now in the hand of naive users include children, older people, and people without the knowledge of network management. Therefore, providing automated OAM for smart homes based on the concept of AIOps is even more important.

Data is a fundamental source of building AIOps based automated solutions. Data is collected during daily operations and also from expensive network equipment in commercial network infrastructures, however, the situation is diverse in smart home networks. Thus, the lack of network traffic data of home networks is the biggest barrier to implement automated OAM solutions for smart homes. In the smart home context, the network traffic data reflects interactions between devices and services utilized these devices, hence it matches to the concept of the IoT Area Network where devices are connected to service gateway(s) (GW) and network traffic data is the traffic between devices and the target service gateway. Generally, network traffic data is generated by real smart homes, testbeds, or simulation. Since network simulation is one of the promising approaches to generate traffic data which achieves the low cost in terms of time and money, a home network simulator which includes (i) a device emulator and (ii) a mechanism to simulate services is proposed as a preliminary component of the network traffic generation framework. Thereafter, a solution to convert the raw network traffic into data used as input for machine learning techniques is introduced to complete the proposed framework. In the scope of the dissertation, the ECHONET Lite protocol which is a dominated protocol for smart homes in Japan is the target protocol for the proposed framework.

The implementation of an ECHONET Lite home gateway (HGW) is one of the first steps to build the simulator. The layered architecture, which includes (i) an adaptation layer to handle the interaction inside of the IoT Area Network, and (ii) an integration layer to integrate with other systems, is proposed. The proposed HGW satisfied all requirements of a GW, as stated in the ITU-T Y.4113 and ITU-T Y.2070, which clarify functional requirements for operation, management, and operation scenarios. To verify the feasibility and reliability of the proposed HGW architecture, the integration with ambient assisted living platform, namely *universAAL*, and a Machine-to-Machine ecosystem, namely *oneM2M*, has been implemented. Experiments have been conducted with the commercial devices, and the results proved the reliability and correct operations of the proposed HGW.

Device simulation is proposed as the last piece of the puzzle of the network simulation because it is hard to have real, controllable faulty devices to generate the faulty traffic which is required for the data set. An ECHONET Lite device emulator has been proposed. The experiment results show that the emulator fully simulated behaviors of real commercial devices. The emulator can simulate faulty devices by extending the concepts of a fault model for the distributed system. Experiment results verified the correct operations of the emulator in simulating normal and faulty devices. By utilizing the Docker platform, automatic and scalable deployment is achievable, and the CPU and memory usage of the device emulator is (**0.15%**) and **100 MB** respectively to simulate a node.

By deploying the network simulator that includes the proposed HGW, normal device emulator, and faulty device emulator, network traffic data is collected in the form of raw captured packets. Data preparation is an essential part that requires to clean the input data and extract meaningful features from data. A network flow calculator, namely *Flowcal*, is proposed to aggregate captured packets into bidirectional flows, which reflect the interaction of devices and the HGW. The *Flowcal* is customized for the IoT area network and is able to appoint the flow initiator and also handle multicast flows. Data samples, which represent device behaviors, are prepared by extracting features from flows and combined into vectors. A total of **91080** samples representing **138** nodes (384 device objects) are extracted from raw captured packets by the proposed solution. The distribution of data samples is visualized by applying the Principal Component Analysis to scale the data dimension. The visualized results prove that **IF-ELSE** is impossible to predict device behaviors.

Since detecting problems is a starting point to build an automated OAM solution and network traffic data could be used to diagnose the health of a smart home network, a network traffic classification application for anomaly detection is implemented based on data generated from the proposed framework to verify the usability of the generated data. Three ML methods include Decision Tree (DT), Support Vector Machine, and Artificial Neural Network (ANN), are investigated for the experiments. All three models achieve high accuracy in classifying normal devices and devices with response fault from the rest of faulty devices. However, the accuracy of detecting omission fault devices and devices with three errors combined is low. The ANN achieves the best performance (average accuracy 96.72%) with data normalization. The DT achieves the best performance (average accuracy 93.23%) without data normalization.

Keywords: IoT Area Network Simulation, **Network Traffic Generation**, **AIOps for Smart Homes**, **Machine Learning Based Network Management**, **Operation, Administration, and Maintenance**

Acknowledgments

Coming to JAIST to undertake this Ph.D. is a life-changing experience for me, and it could be impossible without the support and guidance from many people.

First of all, I would like to express my sincere gratitude to my advisor Professor **Yasuo Tan** of Japan Advanced Institute of Science and Technology (JAIST) for all support and encouragement he gave me during my master course and this Ph.D. course. Without his constant guidance and support with his patient and knowledge, this Ph.D. would not have been achievable. I am proud to be a member in Tan laboratory student.

I wish to offer my special thanks to Associate Professor **Yuto Lim**, my second supervisor, for his valuable comments and advice not only to my research but also about life experiences. He always points out my problems and gives me great suggestions.

I would like to thank Professor **Yoichi Shinoda** for his straight comments as always. Without his comments, I am still an innocent kid in science. Also, I would like to thank Professor **Masao Isshiki** from Kanagawa Institute of Technology his valuable comments and suggestions to improve the quality of this thesis. My examination committee Professor **Razvan Beuran** was instrumental in defining a story for my thesis. For this, I am extremely grateful.

I wish to say sincere thanks to Professor **Satoshi Tojo** and Associate Professor **Nguyen Le Minh**, who taught me and gave me a chance to be at JAIST.

I am grateful to the CARESSES project members, especially Professor **Nak Young Chong** from JAIST and Associate Professor **Antonio Sgorbissa** from the University of Genova for their technical, and financial supports.

I am particularly grateful for the warm welcome and kindly support given by TAN and LIM lab members.

To my beloved wife, I would never complete this Ph.D. without your understanding and tolerance.

Finally, my family is the biggest motivation to go to my studies. My dear parents, your unconditional love made me today. This thesis is dedicated to you.

Contents

List of Figures

List of Tables

Terms and Abbreviations

OAM : Operation, Administration, and Maintenance
AIOps : Artificial Intelligence for IT Operations
IoT : Internet of Thing
ISP : Internet Service Provider
SNMP : Simple Network Management Protocol
CMIP : Common Management Information Protocol
NETCONF : Network Configuration Protocol
ISO : International Organization for Standardization
IETF : Internet Engineering Task Force
FTP : File Transfer Protocol
SMTP : Simple Mail Transfer Protocol
POP3 : Post Office Protocol 3
IMAP : Internet Message Access Protocol
HTTPS : Hypertext Transfer Protocol Secure
HTTP : Hypertext Transfer Protocol
SSH : Secure Shell
DNS : Domain Name System
NTP : Network Time Protocol
P2P : Peer To Peer
M2M : Machine To Machine
PLC : Power Line Communication
RSSI : Received Signal Strength Indication
CAN : Controller Area Network
OSI Model : Open Systems Interconnection Model

Chapter 1

Introduction

1.1 Overview

As stated in the *Theory of Maslow's Hierarchy of Needs* [1], the house provides sheltering and physiological needs of humans and helps humans to maintain their life. Recently, following the development of the IoT, housing-related terms such as *Smart Home*, *Smart Environment*, and *Connected Home* are getting more and more attention to deal with the population aging problems. Smart Homes are spaces utilized advanced information and communication technologies (ICT), which can organize residential infrastructure to improve the overall quality of life (QoL) of residents [2] and assist them to live actively and independently.

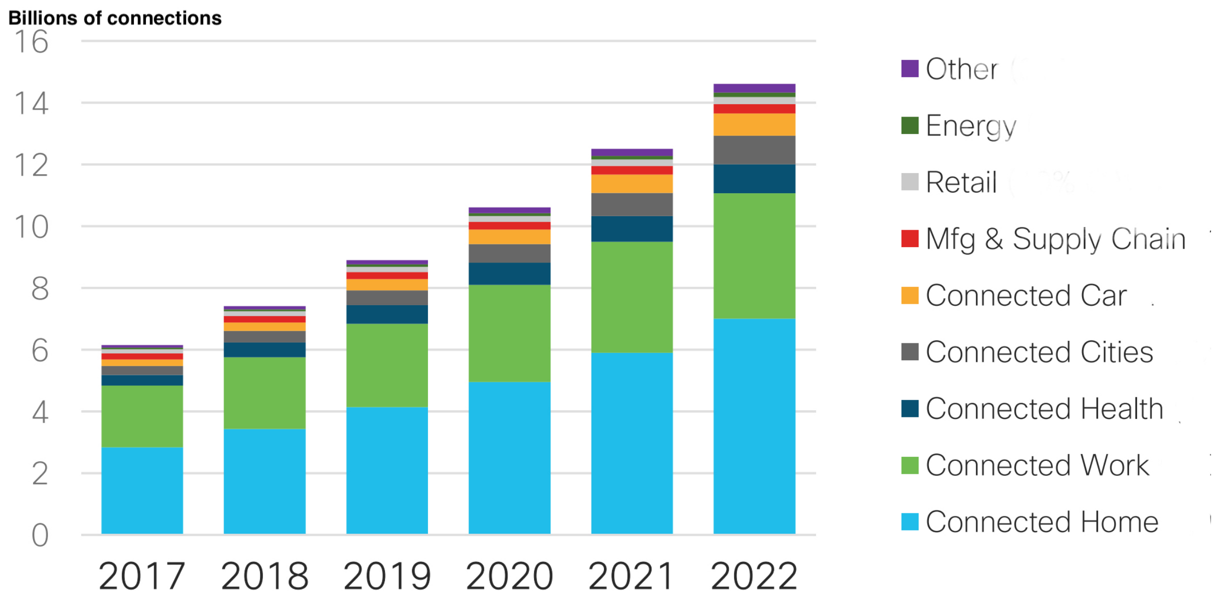


Figure 1.1: Global M2M Connections (Source: *Cisco VNI Global IP Traffic Forecast, 2017 to 2022*)

In the global traffic forecast reported by Cisco [3], the number of connections from smart homes are the largest and increasing steadily. It means that home networks (HN) are complex systems where a large number of heterogeneous devices interconnect with each other and with the global ICT infrastructure. Since users of smart homes are ordinary users without networking expertise, the task of monitoring and managing HN becomes a problematic issue.

Operation, Administration, and Maintenance (OAM) is a term that represents for a group of network management activities that utilizes various tools, network devices, and knowledge to monitor and maintain the stable working condition of networks. As stated in the RFC 6291 [4] and RFC 7276 [5]

- **Operation**: Operation activities are taking to ensure the network (and the services that the network system provides) up and running stably. It includes monitoring the network, finding, and notifying problems. In the ideal case, these problems should be detected before users are affected.
- **Administration**: Administration activities include keeping track of network resources and resource usages. It involves all the necessary bookkeeping to track networking resources and put the network under control.
- **Maintenance**: Maintenance activities mean facilitating repairs and upgrades – for example, when equipment needs to be replaced, or when a router needs to update its operating system image, or when a new switch is installed to a network. The maintenance also includes corrective and preventive measures to make the managed network run more effectively, e.g., adjusting device configuration and parameters.

Generally, these OAM activities are conducted by network operators by gathering information of the network from several sources such as SDN controller, network traffic monitor in order to make decisions to ensure efficient and stable operations of a network. Those decisions are based on know-how that network operators have acquired through their daily operations. However, it is becoming so hard for them to continue such a style of operations in the future, considering the growing volume and kinds of data from several sources. Automated OAM based on concepts of artificial intelligence (AI), and machine learning (ML) which is referred to as *AIOps* (i.e. Artificial Intelligence for IT Operations) is the answer to this problem. The AIOps automates IT operations by utilizing concepts of big data analytic and machine learning to analyze data in order to detect and react to issues automatically.

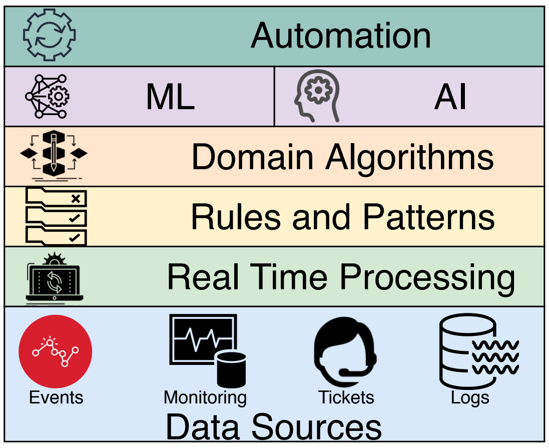


Figure 1.2: The Elements of an AIOps Platform

AIOps concepts leverages existing network data to reduce time consuming manual troubleshooting and analysing so that operators can focus on more high-level work. As in Fig. 1.2, an AIOps platform consists of following elements

- **Data Sources** is the baseline of the platform and data is collected from daily operations such as events, logs, monitoring data, help-desk tickets, etc.
- **Real Time Processing** to processing streaming data based on the concept of big data platform such as Hadoop File system, and Apache Stack, etc.
- **Rules and Patterns** are extracted or discovered from data.
- **Domain Algorithms** is based on domain expertise to interpret extracted rules and patterns in order to achieve goals such as correlating unstructured data, detecting anomalies, and identifying root cause, etc.
- **Machine Learning** and **Artificial Intelligence** to adapt to the new and unknown from the deployment environment.
- **Automation** that leverages outcome from AI and ML to response to identified issues.

Data is a fundamental source of building AIOps based automated solutions. Data is collectable during daily operations and also from expensive network equipment in commercial network infrastructures, however, the situation is diverse in smart home networks. Thus, the lack of network traffic data of home networks is the biggest barrier to implement automated OAM solutions for smart homes.

1.2 Motivation

Unlike a human, machines can monitor the HN 24 hours a day, seven days a week. On the other hand, machines can mimic the human brain to manage the HN as humans learned from experience to get the know-how. Machine Learning (ML) and Deep Learning (DL), which are subsets of AI, can learn from experience like the human brain to perform tasks. The idea is to have an AI model that can monitor and detect abnormal behavior of devices in the HN.

Generative Adversarial Networks (GANs) [6], which was proposed in 2014, is one of the most interesting ideas of the last decade in the field of machine learning. The idea of GANs is derived from the zero-sum game with a **Generator** and a **Discriminator** are countering against the other. The **Generator** is a model that can generate new samples, and the **Discriminator** model tries to classify samples as either real (from the domain) or fake (generated). The two players are trained together in adversarial. Throughout the training process, these two models improve their performances in mutual confrontation and iterative optimization [7] where the **Generator** can generate a sample that is almost the same to the real one and the **Discriminator** can detect every generated sample. In the field of networking, the **Discriminator** can be a network anomaly detector that can detect derived patterns of the network traffic if various patterns of the traffic in the HN were generated as the input.

This work proposes a network traffic generation framework in order to generate smart home network dataset, which is essential to develop ML based solutions to automated OAM tasks of smart homes.

1.3 Main Contributions

This thesis presents processes and solutions to support an AI integrated framework to detect abnormal traffics of devices of smart homes in Japan. To build AI models for AI-based network management, HN traffic data is a must, and the dissertation contributes to the research field of machine learning for HN network management by providing a dataset of traffic in smart home networks. Additionally, the smart home simulator for dataset generation contributes to the field of the smart home simulator by simulating both of the *machine generated traffic*, which is produced by device interaction only and *human generated traffic* which reflects human-environment interactions. The *home gateway*, which is a part of the smart home simulator, is contributing to the interoperability between the European ambient assisted living platform and Japanese smart home protocol. The main contributions are:

1. The HN management framework which supports heterogeneous devices of smart homes;
2. A home gateway architecture which provides interoperability for smart home appliances and service platforms outside of the HN;
3. A device simulator which can simulate normal and faulty network traffic;
4. A smart home simulator which is able to provide both machine-generated and human-generated traffic;
5. A framework to generate network traffic dataset for smart home;
6. A ML-based network traffic classification for anomaly detection.

1.4 Outline

The rest of this dissertation is organized as follows:

- Chapter 1 mainly introduces preliminary concepts, motivation, contribution, and the outline of this dissertation.
- Chapter 2 provides concepts of home networks and home network management. An overview of policy-based network management (PBNM) is described with the advantages and disadvantages of the PBNM are also included. The machine learning-based network management (MLBNM) is briefly described together with a step-by-step process to build MLBNM solutions. Moreover, the state of the art of applying MLBNM solutions for network traffic prediction, fault management, and network security are highlighted . Finally, standardized activities such as standards, protocols, and architectures for network management in the literature review are summarized.

- Chapter 3 describes network models of the smart home environment referred to from the ITU-T Y. 4113 and the target network model in the dissertation, the *IoT Area Network*. Also, the target protocol for the IoT area network, namely *ECHONET Lite*, is described. Finally, the home gateway (HGW) architectures and implementations are introduced. The PoC of the integration of the proposed HGW into an ambient assisted living platform, and a global standards initiative for machine-to-machine will be described.

- Chapter 4 describes the necessity of having a device and network simulators. Furthermore, the device simulation that bases on the target protocol, the ECHONET Lite, is proposed and implemented. Then, the deployment of device emulators and the proposed HGW in Chapter 3 in order to form a home network simulator is described as well.

- Chapter 5 introduce a solution to aggregate raw network traffic into data usable for ML applications. Data preparation process includes a solution to extract bidirectional flows from captured packets for the target network model is proposed. Also, the data processing steps which turn raw input flows into data samples to train ML models are described.

- Chapter 6 mainly explain about building ML solutions to classify the traffic of the target network based on the data set provided in Chapter 5. Furthermore, ML methods include decision tree, support vector machine, and artificial neural network are briefly introduced and investigated. The performance of those methods is highlighted.

- Finally, chapter 7 summaries results and outlines future research directions.

Chapter 2

Background and State of the Art

2.1 Background

2.1.1 Home Network

As stated in [8], HN is

> **A short-range communications system designed for the residential environment, in which two or more devices exchange information under some sort of standard control.**

A typical HN has grown from a simple network with a few devices such as telephones and computers to a complex system that connects HVAC devices, digital entertainment devices, home security, smart appliances, sensors, and actuators into a common network. Today HN is a complex system that allows devices to communicate with each other to share resources and also to service providers outside of the HN via the Internet. Devices in the HN are heterogeneous IoT devices due to they are different in terms of functionalities, processing power, communication capabilities. There are two main trends in current smart home appliances:

- **Device-Cloud Oriented** (DCO): Devices directly connect and use the services provided from cloud or servers, which are outside of the HN via the Internet. Amazon Echo is one example of this pattern. Amazon Echo is a smart speaker provided by Amazon, and it connects to the Amazon cloud, namely *Alexa*. Amazon Echo takes user's voices as input and sends it to the cloud server, and requests are processed at the server then results are sent back to the Amazon Echo.

- **Service Platform Oriented** (SPO): Devices such as sensors and actuators are connected to a device called **Home Gateway** (HGW) that is inside of the HN. The HGW acts as a man-in-the-middle or a proxy between these devices and service providers from outside of the HN. In general, these devices can be remotely accessed via the HGW. However, they only connect to the HGW via the local area network.

The overview of **Device-Cloud Oriented** architecture and **Service Platform Oriented** architecture are illustrated in Figure. 2.1.

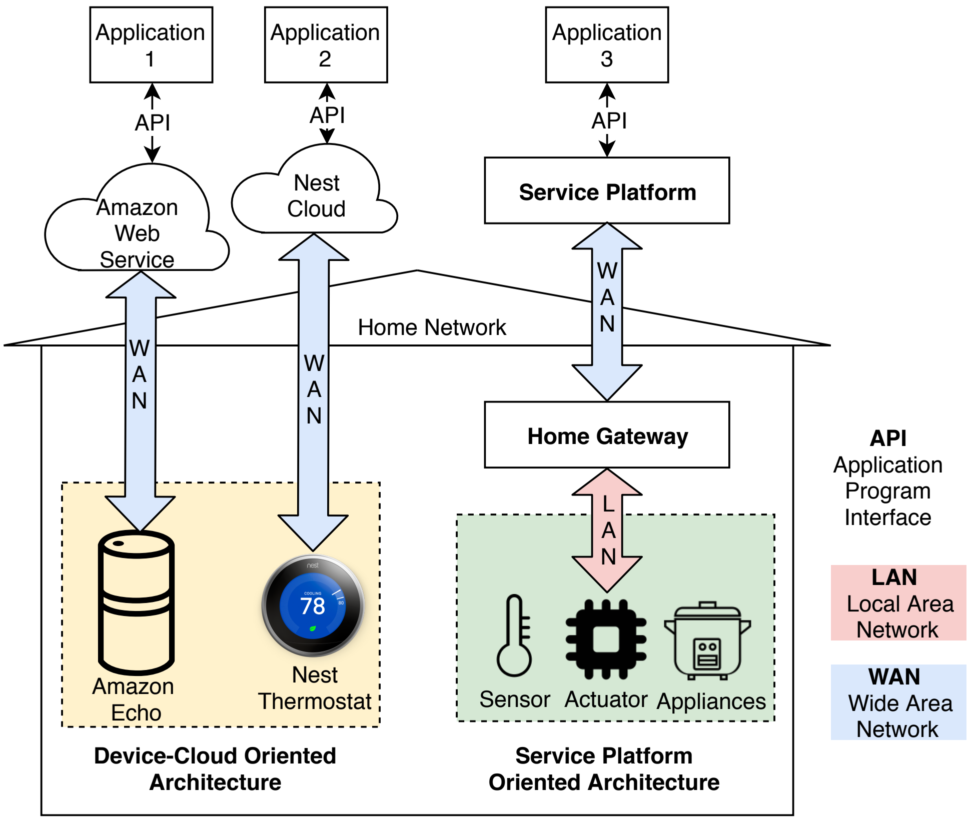


Figure 2.1: Device-Cloud Oriented and Service Platform Oriented Architecture in Smart Homes

2.1.2 Network Management

According to the ISO, functional requirements of a network management model includes:

- *Fault Management* to support fault detection, notification, and correction to keep the network running steadily and stably. Since faults can create downtime and unacceptable network degradation, fault management is essential and the most widely implemented element of the ISO model [9].
- *Configuration Management* to track, monitor, and manage configuration information of network systems.
- *Performance Management* to track and audit various aspects of performance so as to maintain the acceptable level of network system overall performance.
- *Security Management* to monitor access to network resources in order to prevent network sabotage, abuse, and unauthorized access.
- *Accounting Management* to manage the usage information of network resources.

In general, most network management architectures share a standard structure and set of relationships, as shown in Figure. 2.2. A typical architecture has:

- A *Management Agent*, which resides in an end device such as computers or networked devices, is a software to monitor and report the status of the end devices and also execute instructions from the network manager(s).

- A *Management Entity*, which acts as a network manager, collects information from **Managed Agent(s)** and provides instructions for **Management Agent(s)** to maintain the stable operation of end devices. The *Management Entity* and the *Management Agent(s)* communicates with each other via network management protocol such as SNMP, NETCONF, or CMIP, etc.
- A *Management Proxy* is an *Managed Agent* that provides management information on behalf of other entities.

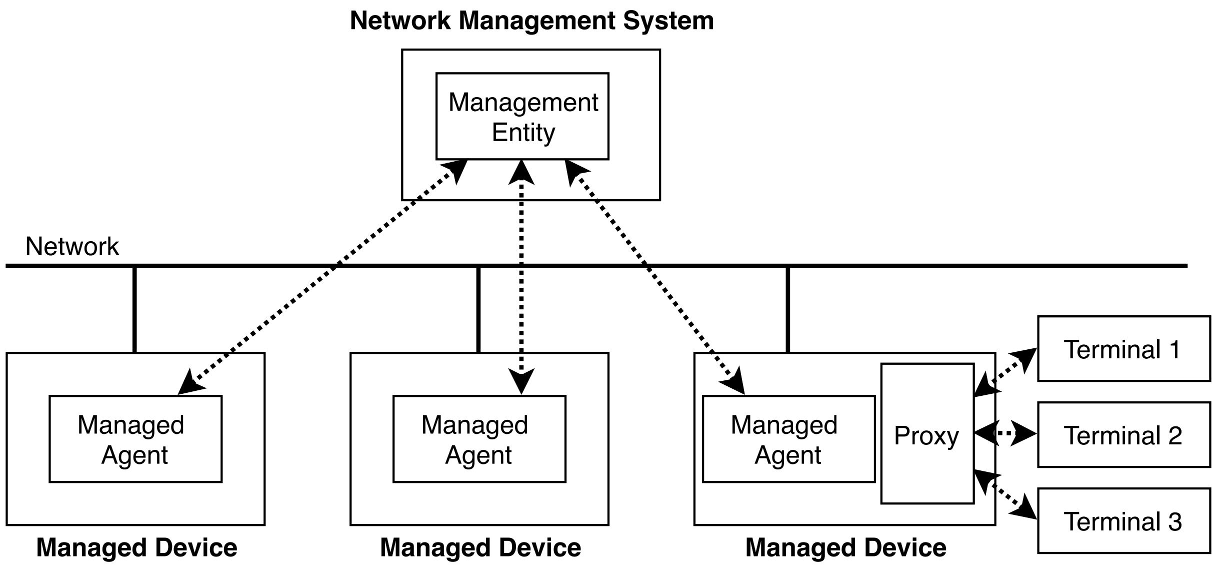


Figure 2.2: A Typical Network Management Architecture

2.1.3 Policy-based Network Management

Policy-based Network Management (PBNM) had widespread attention after the IETF released the *Policy-based Network Management Architecture* as in Figure. 2.3. The architecture was considered as the best solution for the Internet policy-based management which consists of (i) the **Policy Console** (PC) to specify policies by supporting create, modify or remove policy abilities; (ii) the **Policy Repository** to store policies created by the PC; (iii) **Policy Decision Point** to make decision, translate and transform policies and enforce policies; and (iv) the **Policy Enforcement Point** to apply policies including verifying, executing and validating the execution of policies. The term **Policy** refers to rules containing (i) *Event* (when the policy will be triggered); (ii) *Action* (task(s) needs to be done in the scope of the policy);(iii) *Condition* (criteria for triggering a policy); and (iv) *Priority*.

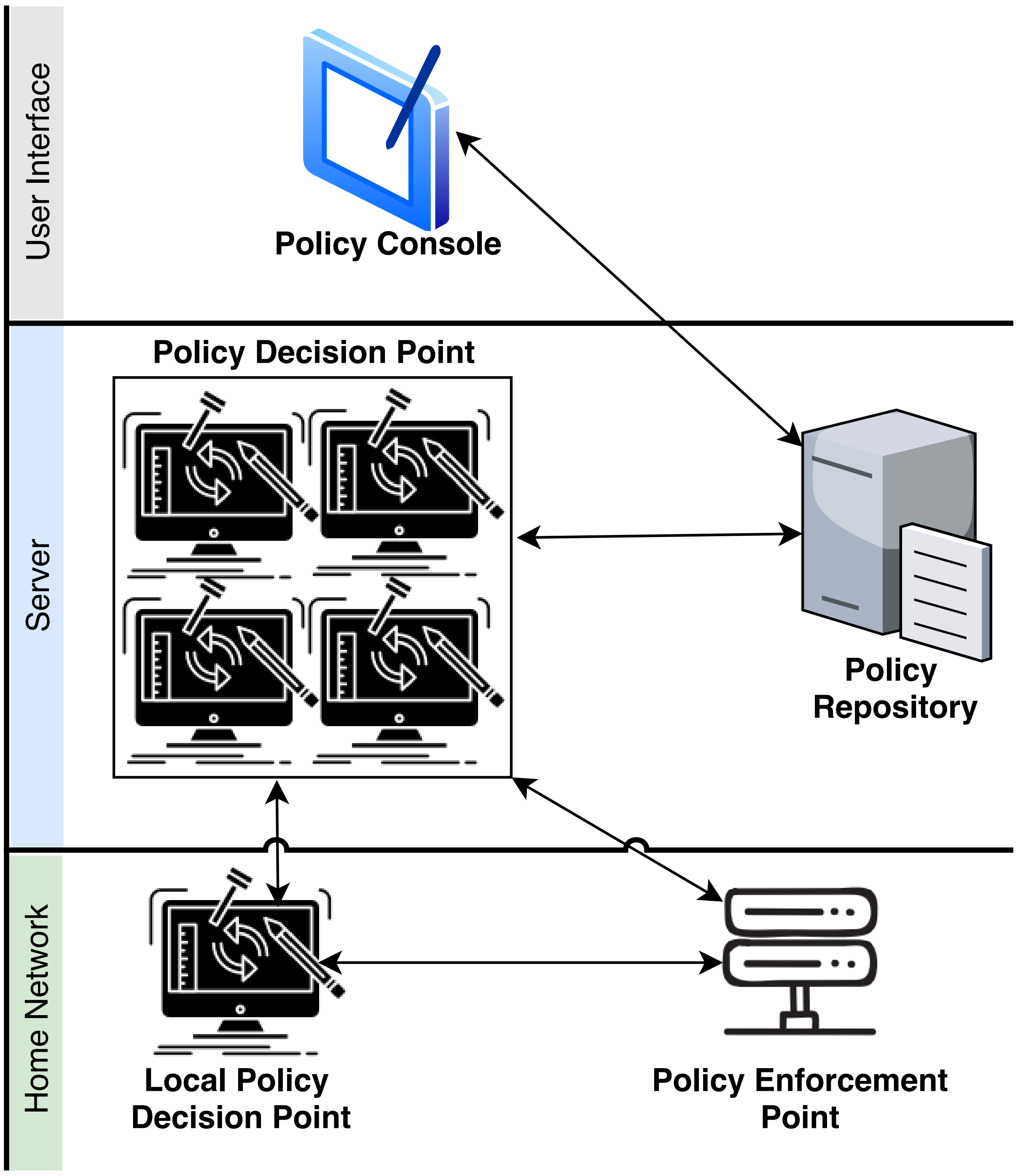


Figure 2.3: Policy-based Network Management Architecture

The benefit of the PBNM is that it is easy to achieve the automation of network management. The administrator can interact with the network via human-friendly policies [10]. Additionally, it can adapt to the change quickly via run-time reconfiguration of policies. However, rules-based systems are limited since they must follow the predefined rules and can not handle subtle cases. The PBNM requires effort from experts to update policies accordingly, and it is impossible for smart homes.

2.1.4 ML-based Network Management

The ML evolution has achieved breakthroughs in several domains such as computer vision, speech recognition, self-driving cars, and network management [11]. Unlike PBNM, where computers can tirelessly and consistently perform repetitive and well-defined policies provided by the network operators, machine learning-based Network Management (MBNM) can create policies based on the situations of the HN by learning from network operator perspectives [12].

A typical flow of applying ML in networking is as in Figure. 2.4.

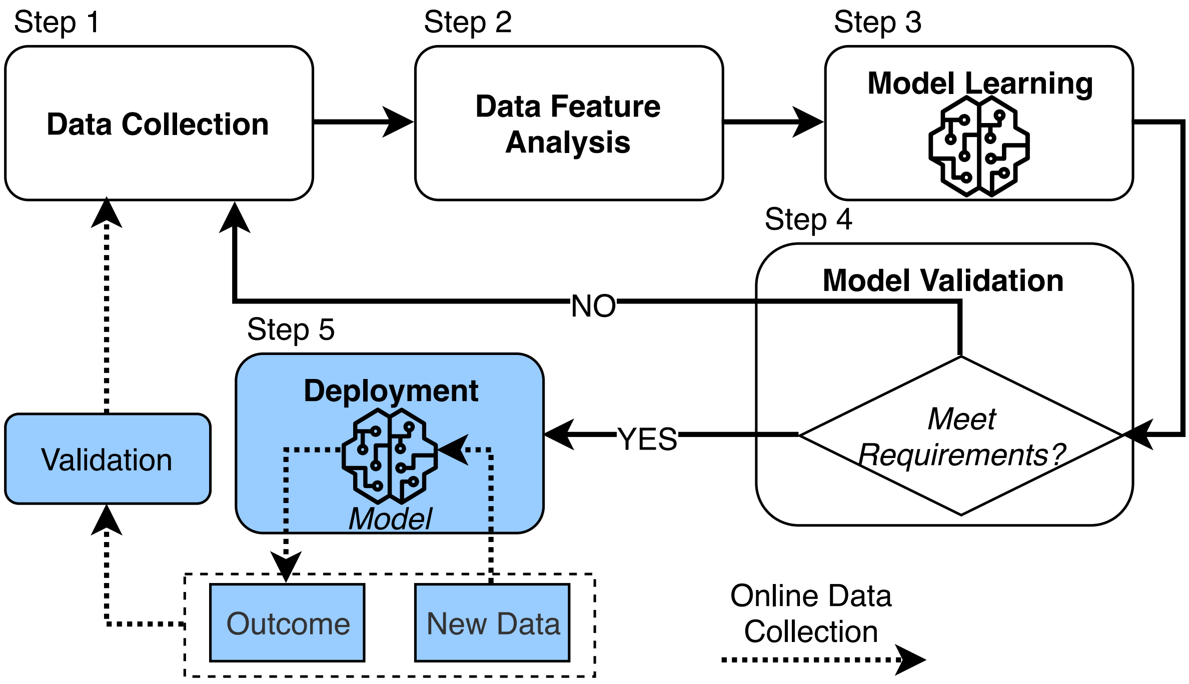


Figure 2.4: A Baseline Workflow of Machine Learning for Networking

For a given problem, ML requires data, possibly without bias, to build the ML model. Therefore, **Data Collection** is an important initial step [13]. Network data such as network traffic, logs from network devices can be collected by deploying real network laboratory, emulated environment, or synthetic environment [14]. Data can be collected in two phases: online and offline [15]. In the offline phase, a large amount of historical data is used for model training and testing. Data collected from the online phase is used as feedback to re-train the model. **Data Feature Analysis** is an important step before training and testing the ML model. Since each problem has its characteristics and only several factors from the raw collected data are meaningful for that specific problem, finding and choosing the proper features is one of the keys to fully unleashing the potential of data. **Model Learning** is a process that includes model selection, training, and adjusting. A suitable model or algorithm can be selected according to the problem category, size of data, requirements, and so on. **Model Validation** evaluates whether the chosen model works well or not. Usually, the overall accuracy of the model is used for cross-validation. The predefined steps might need to be repeated if the requirements have not been met. **Deployment** is the last step when the trained model has been deployed into practical applications that take real-time input and produce corresponding output.

2.2 State of the Art

2.2.1 Machine Learning for Networking

Traffic Prediction

Traffic prediction provides forecasting on future traffic and supports network planning, resource provisioning, optimization, and so on. Table. 2.1 summaries research activities on applying ML for traffic prediction.

Table 2.1: Machine Learning for Network Traffic Prediction

	ML Technique	**Application**	**Dataset / Features**	**Output**
Network Bandwidth Predictor[16]	Supervised Learning Multi-Layer Perceptron Neural Network	End-to-end bandwidth availability prediction	NSF Teragrid Dataset/ Min, Max, and Average load	Available bandwidth in future
Internet Traffic Forecasting[17]	Supervised Learning Neural Network with Resilient Propagation	Link load and traffic volume prediction in ISP networks	SNMP traffic data from 2 ISP networks/ Traffic volume observed	Expected traffic volume
Link Load Prediction [18]	Supervised Learning Support Vector Regression	Link load prediction in ISP networks	Internet traffic of an ISP network/ Link load observed	Expected link load
Internet Traffic Prediction [19]	Supervised Learning Multi-Layer Perceptron Neural Network	Network traffic prediction	1000 points dataset/ Past measurements	Expected traffic volume
Network Traffic Prediction [20]	Supervised Learning Particle Swarm Back Propagation Neural Network	Network traffic prediction	2-weeks of houly traffic/ N past day houly traffic	Expected next day hourly traffic volume
Data Center Network Traffic Prediction[21]	Supervised Learning Multi-Layer Perceptron Neural Network	Data Center traffic volume prediction	6 weeks of traffic from Baidu/ Time and frequency from total and elephant traffic	Expected traffic volume of future 30s blocks
Future Traffic Prediction[22]	Supervised Learning Long Short Term Memory	Future traffic inferring based on flows	24 weeks of network traffic & flow count/ Flow count	Expected traffic volume
Online Flow Size Prediction[23]	Supervised Learning Multi Layer Perceptron Neural Network	Early low size prediction and elephant flow detection	3 campus networks and 3 million flows each/ Netflow, size of 3 first packets	Flow size classes elephant flow vs non elephant flow prediction

Fault Management

Table 2.2: Machine Learning for Fault Management

	Application/ ML Technique	**Target Network/ Dataset**	**Features**	**Output**
Proactive Network Fault Prediction[24]	Fault Prediction/ Bayesian Network	Campus Network/ Data from router	MIB based information includesInterface, IP, UDP	Network health prediction
Cellular Network Fault Prediction[25]		Cellular Network/ Simulation	Power, Cell Transmission, Multiplexer	Faulty or Not
Dependability of Wireless Network[26]	Fault Prediction/ Neural Network	Wireless Network/ Simulation	Mean time to failure/ restore Time Profile Run Time	Dependability, Survivability, Availability, Failed component
Link Quality Prediction[27]	Fault Prediction/ Support Vector Machine, Bayesian Network	Wireless Sensor Network / Sensor Network Testbed	RSSI, buffer size, Chanel load	Link quality estimation
Autonomic Failure Prediction[28]	Fault Prediction/ Hessian Locally Linear Embedding	Distributed System/ File Transfer Application Testbed	Performance, IP, TCP, UDP	Network, CPU, Memory failure prediction
Optical Network Failure Prediction[29]	Fault Prediction/ Support Vector Machine	Optical Network/ Telecommunication operator data	Input/ output optical power Temperature Unuseable time	Equipment failure prediction
Operational Fault Detection[30]	Fault Detection/ Statistical learning	Cellular Network/ Data from real cellular networks	Mobile user call load profile	Base station, sector, carrier, and channel fault detection
Network Traffic Fault Classification [31]	Fault Detection/ K-mean, Fuzzy C Mean	Campus Network/ Network with heavy and light traffic	SNMP data	Normal/ link failure traffic, server crash, broadcast storm classification
Network Failure Logs Mining [32]		Service Provider Network/ data from service providers	Fault occurence time, Geographical, fault cause, resolutiontime	Identify spatio temporal patterns
Adaptive Fault Detection[33]	Fault Detection/ Change Detection Method	Local Area Network/ Real network	SNMP Data	Detect anomaly as soon as possible

Fault management is a headache problem for network operators and administrators since it requires a thorough knowledge of the entire network and the running application of the network. Table 2.2 summaries ML-based fault management approaches. However, the biggest challenge is that it lacks data in the production network as it is difficult to generate faulty data in the production network. Injecting faults could be a solution [34], however, it is unrealistic to inject faults into a production network for data generation purposes. Therefore, unsupervised learning and reinforcement learning techniques could be potential solutions for fault management.

Network Security

Network security purpose is to protect networks from threats where attackers always find clever ways to attach networks [35]. ML has been widely applied in cybersecurity and intrusion detection [36]. In general, ML improves the rule-based systems and can detect complex patterns of attacks. However, the **KDD'99**[37], which is an out-dated dataset, has been used for the majority of researches. Therefore, it does not reflect the recent types of attacks.

Summary

ML has been applied successfully in networking. It shows promising results to provide OAM services for networking. The critical success of ML is the availability of data, which is achievable by the explosive growth inf traffic volume and connected devices. However, current research works focus on the core network and the Device-Cloud Oriented architecture in the home network (Figure. 2.1). There is no literature about the DCO architecture in the HN so far since it lacks datasets for the SPO architecture for several reasons:

- Devices are constrained, and they can not have a monitoring agent.
- Devices are heterogeneous in terms of communication medium.
- It lacks a standardized communication protocol between devices and the HGW.

2.2.2 Standardization Activities in Home Network

ITU-T Y.2070

ITU-T Y.2070 [38] is a recommendation that focuses on **Next Generation Networks -Frameworks and functional architecture models**. The ITU-T Y.2070 specifies the requirements and architecture of the home network services and is widely referred to in the research field of HN. The recommended HN service architecture is shown in Figure. 2.5. The recommended HN architecture focuses on the SPO architecture that includes:

- **Requirements for device**: it is required for devices to have an abstract data model to reflect device operation. Moreover, it is required to have a *managed agent* that provides information to the manager.
- **Requirements for Home Gateway** (HGW): It is required to support protocol conversion to deliver information from LAN to WAN to the management platform. For management purposes, it is required to support the concept of a resource information collector to discover new devices and retrieve connected device statuses. Additionally, it is required to have a local management mechanism in case of network disconnection.

- **Requirement for Management Platform** (MP): it is required to support the concept of virtual devices, which are a logical mapping of physical devices. Additionally, it is required to support API to allow services to access HN resources. For management purposes, it is required to discover and control devices connected to the HGW remotely from WAN.

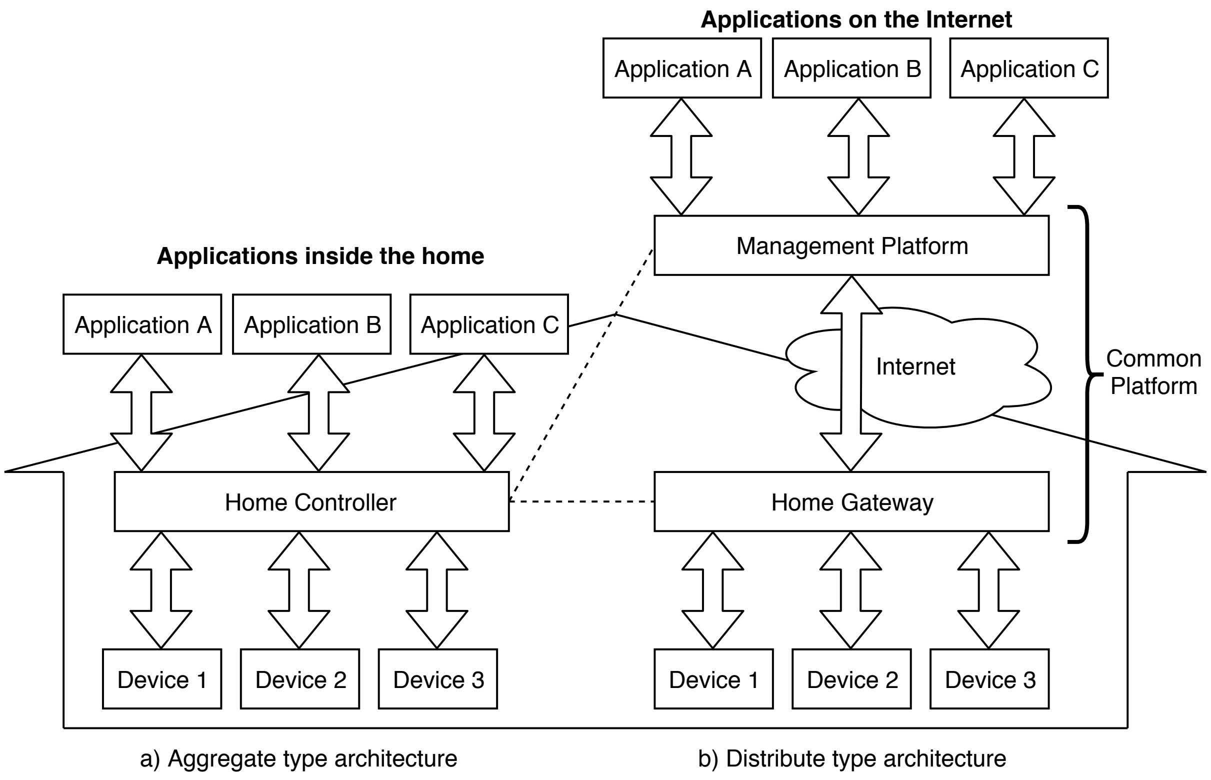


Figure 2.5: ITU-T Y.2070 Home Network Service Architecture

OmniRAN (802.1 CF)

Modern networks such as smart homes, smart grid, and smart cities are heterogeneous, which support multiple network interfaces and various network access technologies. Therefore, omniRAN scope is to unify the support of different interfaces, enabling shared network control and the use of software-defined network (SDN) principles, thereby lowering the barriers of heterogeneous IEEE 801 families. OmniRAN stands for

- **RAN**: Range Area Networks
- **Omni**: Open Mobile Network Interface

which is a recommended practice for the network reference model and functional description of the IEEE 802 access network. The OmniRAN network architecture is illustrated in Figure. 2.6.

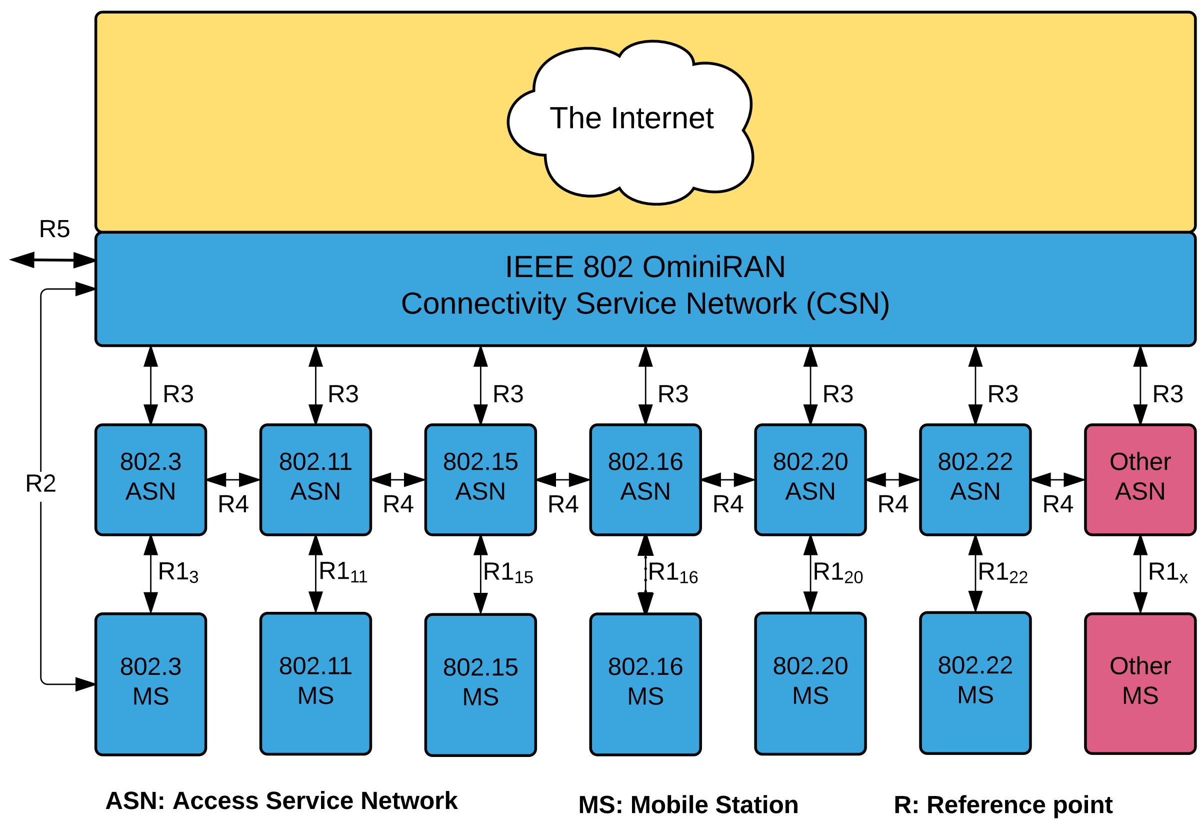


Figure 2.6: Overview of OmniRAN Architecture

The *Connectivity Service Network* (CNS) is the core functionality of the OmniRAN which provides a common interface for IEEE 802 access technologies. The *Access Service Network* (ASN) is unified network interfaces customized for each technology.

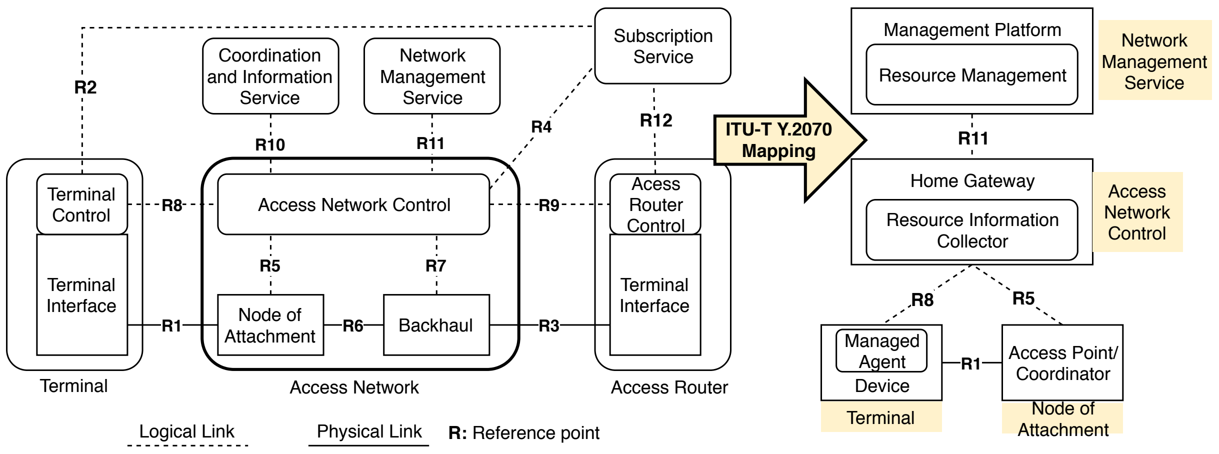


Figure 2.7: OmniRAN Network Reference Model and ITU-T Y.2070 Mapping

The OmniRAN network reference model (NRM) includes

- **Terminal** is an end device which connects to a node of attachment of the access network via wired or wireless interface. A node of attachment is either an access point or base station.

- The **Access Network** aggregates and forwards traffic from terminal to the access router via the backhaul.
- **Access Router** terminates the layer-2 link from the terminal and forwards user traffic to information servers according to IP addresses in the payload of the layer 2 data frames.
- Services are able to provide to the terminal directly via the agent resides inside the terminal or indirectly via the **Access Network Control**

The OmniRAN NRM can be mapped into the ITU-T Y.2070 which specifies the service architecture of home networks as in Figure. 2.7. The OAM is also supported in the OmniRAN by being described in fault diagnostic and management service.

Home Network Topology Identifying Protocol

Home Network Topology Identifying Protocol (HTIP), [39] is specially designed for HN. HTIP defines a protocol to check the connectivity of devices and the overall topology of an HN. HTIP covers a wide range of end devices such as PCs, TV, gaming devices, printers, sensors, and so on. Moreover, HTIP supports heterogeneous transmission medium, especially in the HN, such as PLC, wired, wireless, CoAx, and UTP cable [40]. The overview of HTIP is illustrated in Figure. 2.8.

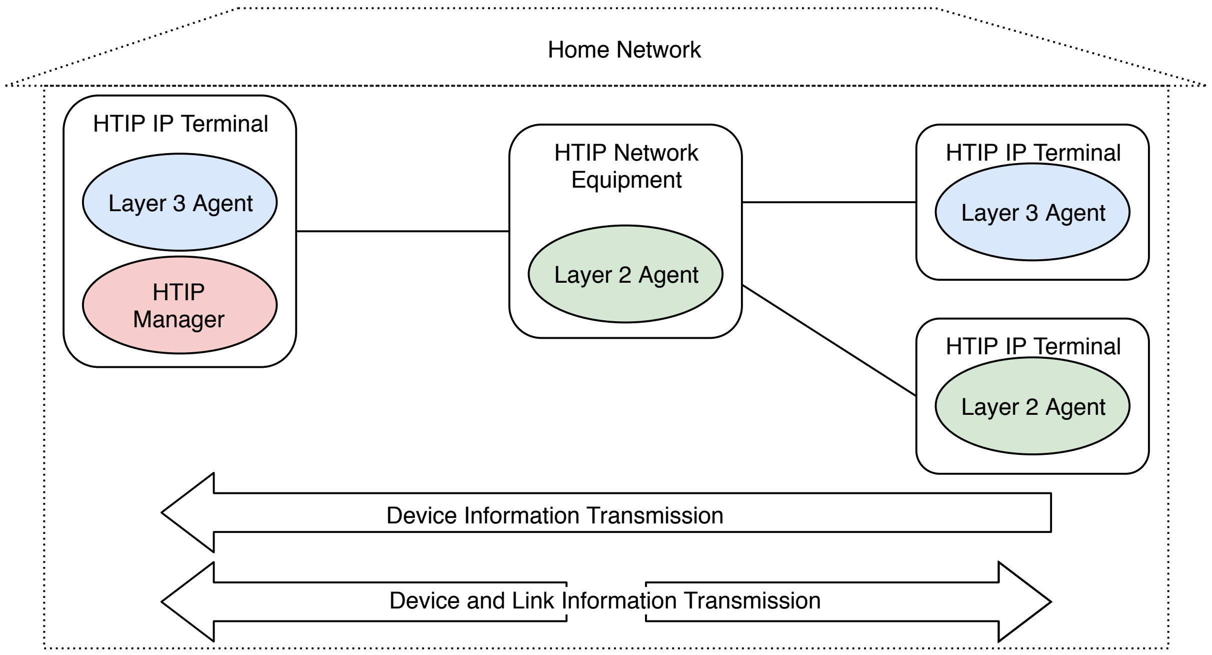


Figure 2.8: Overview of HTIP Architecture

HTIP supports managed agent (as in Figure. 2.2) for lightweight home appliances. HTIP agents can be installed inside devices (terminals) or network devices (switch, router, access point, etc.). The HTIP manager is responsible for collecting information from agents. There are two types of information:

- **Device Information** consists of (i) Device category, (ii) Manufacturer code, (iii) Model name, and (iv) Model number as mandatory attributes. The device information is useful information to identify devices in the HN.
- **Link Information** is only collected from HTIP network equipment. The link information provides information on pairs of *[Port, MAC address]*, which is usable to identify topology on HNs. Moreover, the link information may include useful information for fault detection and isolation in HNs such as *Pairing information, channel usage information, RSSI, etc.*

The overall HN topology of an HN provided by the HTIP is useful for HN fault detection, isolation, and recovery [41].

2.3 Summary

HN is becoming more and more complicated due to the rapidly growing of connections inside the HN. Network management for HNs is more complicated, and the traditional way of network management using **Policy-based** approaches seem challenging to adapt to the heterogeneous of devices in HNs. Recently, ML technologies achieved breakthroughs in various fields, including networking. The ML-based approach shows compromising results. It reduces human effort on network management widely applied in core networks, campus networks where devices and equipment support management protocols such as SNMP and SDN. However, there has been no effort in HN management since it lacks a dataset of HN. Data is a fundamental source of building ML-based solutions for HN management. The basic workflow of data collection is as in Figure. 2.9

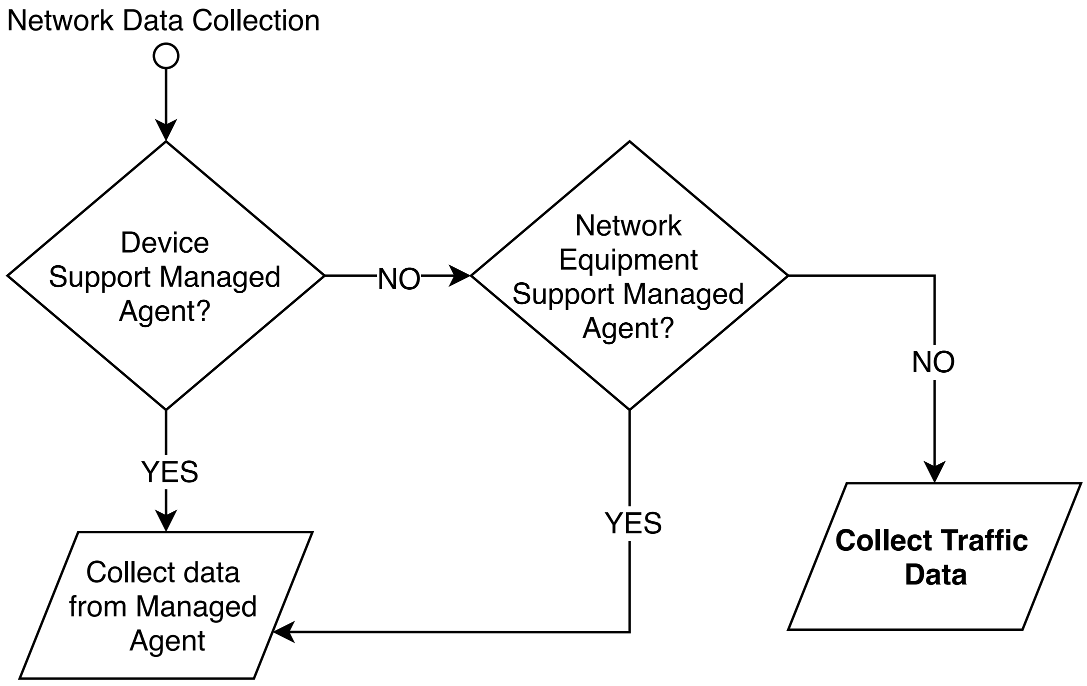

Network Data Collection
Device Support Managed Agent?
NO
Network Equipment Support Managed Agent?
NO
YES
YES
Collect data from Managed Agent
Collect Traffic Data

Figure 2.9: Data Collection Flows in Home Networks

In general, in the HN, where devices are constraint thus they do not support a managed agent, and network equipment is low-cost equipment, and it does not support a management protocol such as SNMP or even HTIP. Therefore the only way to collect information for management purposes is via the traffic of devices during operations.

Chapter 3

Service Gateway Architecture for Service Simulations

3.1 Target Home Network: IoT Area Network

The basic model of the IoT network (in the context of smart homes) that refers from the ITU-T Y.4113 [42] is illustrated in Figure. 3.1.

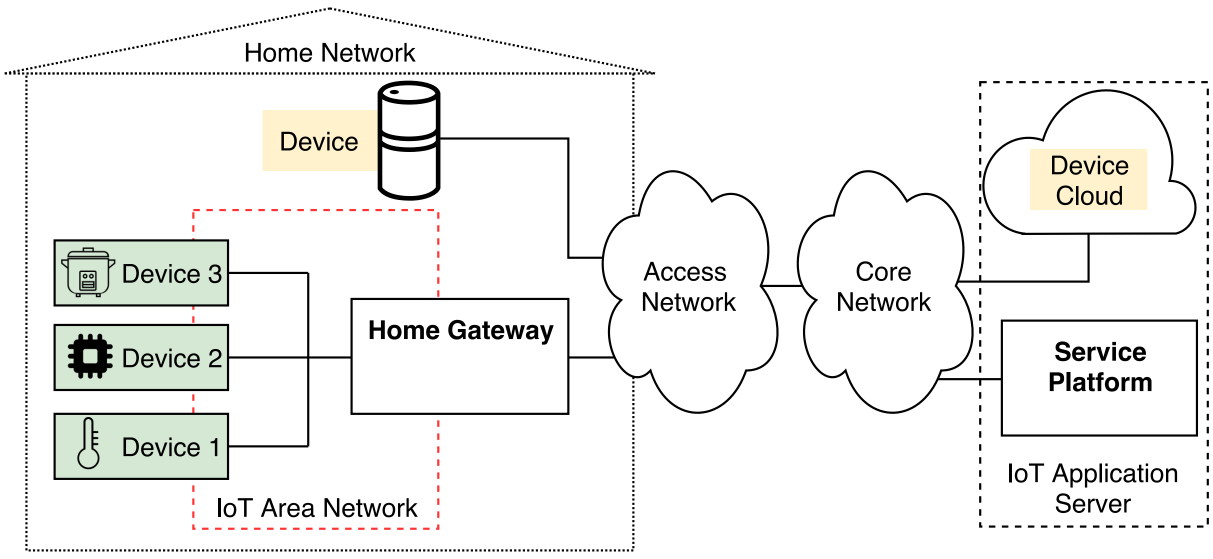


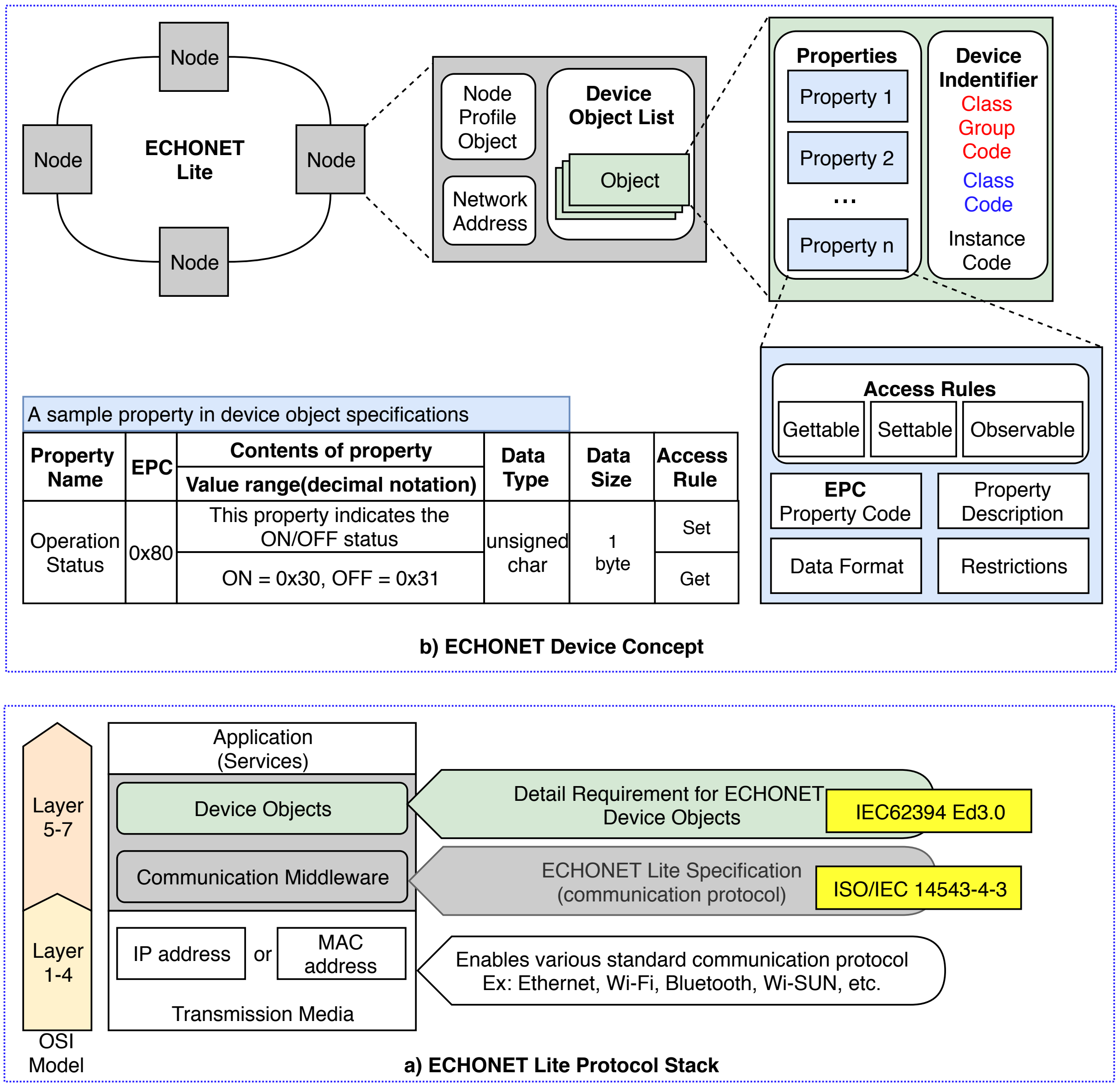
Figure 3.1: Basic Model of IoT Network in the context of Smart Homes

The gateway is responsible for interconnecting devices with the core network. It includes a proxy translation between device protocols and core network protocols and device resources and services management. The core network provides communication infrastructures to connect service providers and access networks. The access network connects devices and gateway(s) to the core network. The IoT area network is a local area network that interconnects devices and gateways by utilizing short-range communication technologies such as CAN, Zigbee, Bluetooth, Wi-Fi, Wi-SUN, and so on. As the access network and core network are managed by ISPs, this dissertation targets on providing OAM services for the **IoT area network**.

In [43], a management architecture that supports direct management techniques for

DCO devices and indirect management for SPO devices (Figure. 2.1) via an intelligence HGW. The proposed solution [43] proved that an intelligence HGW could manage the IoT area network efficiently and also guarantee the interoperability for the smart homes. However, the proposed intelligence HGW was implemented by the policy-based approach (Section 2.1.3), which requires knowledge of experts for policy definitions. By employing the ML-based solutions, the limitation of the previous might be resolved.

3.1.1 Target Home Network Protocol: The ECHONET Lite



A sample property in device object specifications

Property Name	EPC	Contents of property	Data Type	Data Size	Access Rule
		Value range(decimal notation)			
Operation Status	0x80	This property indicates the ON/OFF status	unsigned char	1 byte	Set
		ON = 0x30, OFF = 0x31			Get

Figure 3.2: Concepts of the ECHONET Lite protocol

ECHONET, which has become a de facto home network standard certified by ICE and ISO, stands for Energy Conservation and Homecare Network. However, the ECHONET protocol did not attain widespread adoption due to two significant factors. Firstly, the specification requires a more complicated system configuration for multiple controllers and multiple devices. The other factor was the overall complexity of the protocol, leading to only a few compliant implementations. Therefore, in 2011, it was redesigned as the substantially simplified ECHONET Lite protocol.

ECHONET Lite has become a leading interface used in smart homes in Japan, and the number of devices compatible with the ECHONET Lite protocol is growing steadily [44]. The basic concepts of the ECHONET Lite protocol are illustrated in 3.2 where a network of ECHONET Lite devices is a collection of *Nodes*. A node is a physical device connected to the network. Each node contains the *Network Address* and *Profile Object* which identify a node, and a list of *Device Objects*. A device object represents a logical device that is classified into seven groups and 108 classes of devices in the latest specification released in 2017 [45]. Device objects offer a standardized method to represent device resources and services via a list of *Properties* and corresponding constraints for each property.

As shown in Figure. 3.2 **a)**, the ECHONET Lite protocol stack defines layer five to seven of the OSI model. The lower layer is left free in order to support IoT area network technologies. Furthermore, the ECHONET Lite protocol provides the concept of *device object* (Figure. 3.2 **b)**) which is a requirement for IoT devices in HN. Since the ECHONET Lite protocol is able to cover the concept of IoT area network (as specified in Section 3.1), ECHONET Lite protocol is the target protocol in this dissertation.

3.2 Home Gateway Architecture

Table 3.1: Requirements for Home Gateway (referred from ITU-T Y.2070)

Requirements	**Functions**	**Description**
Device Operation	Data Format & Protocol Conversion	To convert the communication protocol of home network to HTTP protocol for sending information of connected physical devices to the management platform (outside of the house). Moreover, it is required to convert data format of physical devices into a common data modelusable by others to achieve interoperability
Management	Resource Information Collector	To collect information from devices includes discover, activate, monitor and control connected devices. Moreover, it is required torecognize newly connnected devices and generate a unique identifier for each devices
Application Execution	Application for disconect	To autonomously and locally monitor devices and backup data in case of network disconnection

Table 3.1 summaries requirements for a HGW. Since devices in IoT area networks must connect the HGW and the HGW is a central point that manages and represents for them, the traffic between HGW and devices are usable to diagnostic the health of IoT area networks. The mapping between requirements and components of the HGW in the context of HN is visualized in Figure. 3.3.

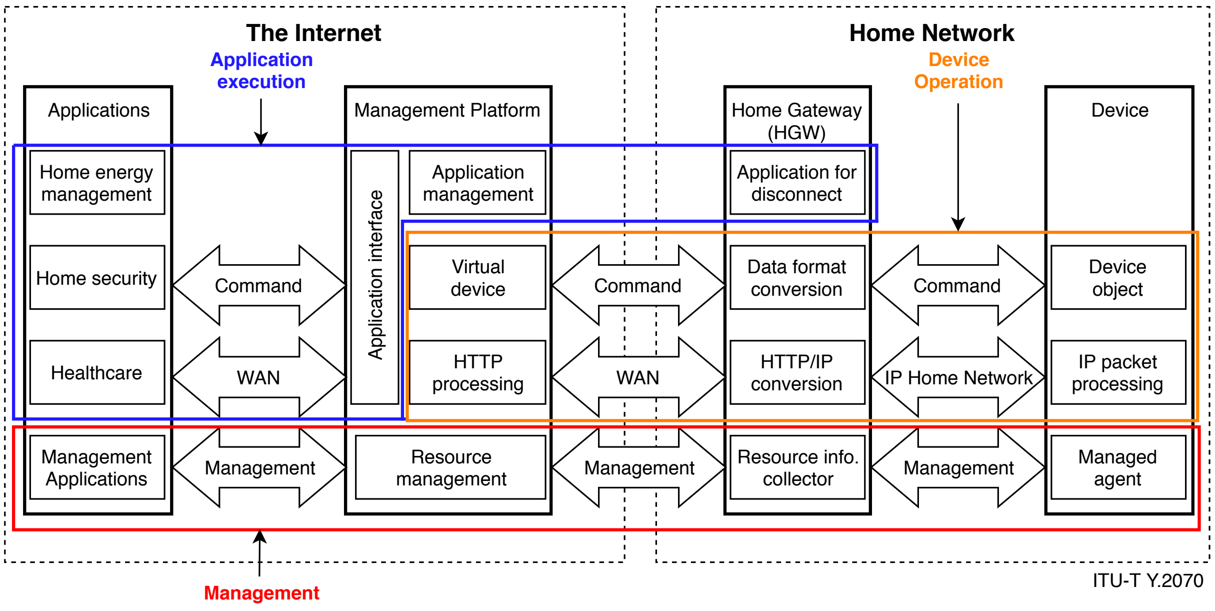


Figure 3.3: Overview Functional Architecture of a Home Gateway

To support the flexible and extensible implementation and integration with other systems, the proposed HGW architecture is divided into:

- **Adaptation Layer** is responsible for monitoring devices and transform device resources and services into a common data model includes
  - **Frame Translator** acts as a translation proxy to interact with devices using packets.
  - **Network Monitor** covers the requirements of the *Resource Information Collector* that continuously monitors device resources by sending commands and parsing the response into device objects.
  - **Data Exporter** supports a common data model and mechanisms to transform device objects into data usable by other systems.
- **Integration Layer** supports the *Application Execution* requirements and provides APIs for the *Management Platform*.

Since ECHONET Lite has been chosen as the target protocol for the evaluation, implementations of the proposed HGW architecture are based on the ECHONET Lite protocol.

3.3 Home Gateway Adaptation Layer

3.3.1 Related Work

A list of ECHONET Lite SDKs referenced from Smart House Research center of Kanagawa Institute of Technology [1] is summarized in Table. 3.2. These SDKs provide tools and middleware which basically implemented frame translator functions and device interaction

[1]http://sh-center.org/en/

APIs to support sending and receiving frames in order to control ECHONET Lite devices in a simple manner. However, mechanisms to manage a network of devices have not been stated as well as the interoperability issues have not been mentioned.

Table 3.2: ECHONET Lite SDK

Name	**Category**	**OS**	**Programming Language**
SSNG	Tool	Window	-
SSNG for iPhone [46]	Tool	iOS	-
SSNG for Node JS	Tool	Window MacOS Linux	Node.js
echonet-lite	Middleware	-	JavaScript
library for Swift 3.0	Middleware	iOS	-
OpenECHO	Middleware	-	Java
Temperature Sensor EL	Emulator	iOS	-
Light Emulator -1	Emulator	Window MacOS Linux	Java

In [47], a proxy that integrated ECHONET base smart home to PUCC [2] protocol has been proposed. This implementation achieved interoperability at the communication level, which was able to facilitate ECHONET Lite devices to service platforms (outside of the home network). However,the interoperability at the data level has not been introduced.

There was another effort from the ECHONET consortium[3] to map from the concept of device objects into a smart device template (SDT) [48] to gain interoperability for ECHONET Lite devices. The SDT aims to provide an abstraction layer to describe appliance resources in XML format. XML is intuitively manifest for human beings because tag names are able to provide semantic meaning. However, machines do not work that way due to the lack of intuition, and consequently, the usage of XML for semantic interoperability will be ineffective in the long run, as stated in [49]. Therefore, the ontology model for ECHONET Lite specification is still desired.

In [50], the SAREF ontology was proposed for the smart appliances domain. An overview of SAREF is illustrated in Figure. 3.4. SAREF is not intended to replace existing standards but to provide a domain abstraction model to link information from different technologies and domains for semantic interoperability based on ontology. The SAREF ontology becomes a part of ETSI [4] standards [51]. There have been many extensions of SAREF to support ontology models for environmental devices, smart building devices, smart city infrastructure, and energy domain. There is no such model for ECHONET

[2]http://pucc.jp/
[3]https://github.com/ECHONET-Consortium/ECHONET-SDT-Contribution
[4]https://www.etsi.org/

Lite devices. Thus an ontology model that is able to extend to SAREF is able to improve the semantic interoperability for ECHONET Lite protocol.

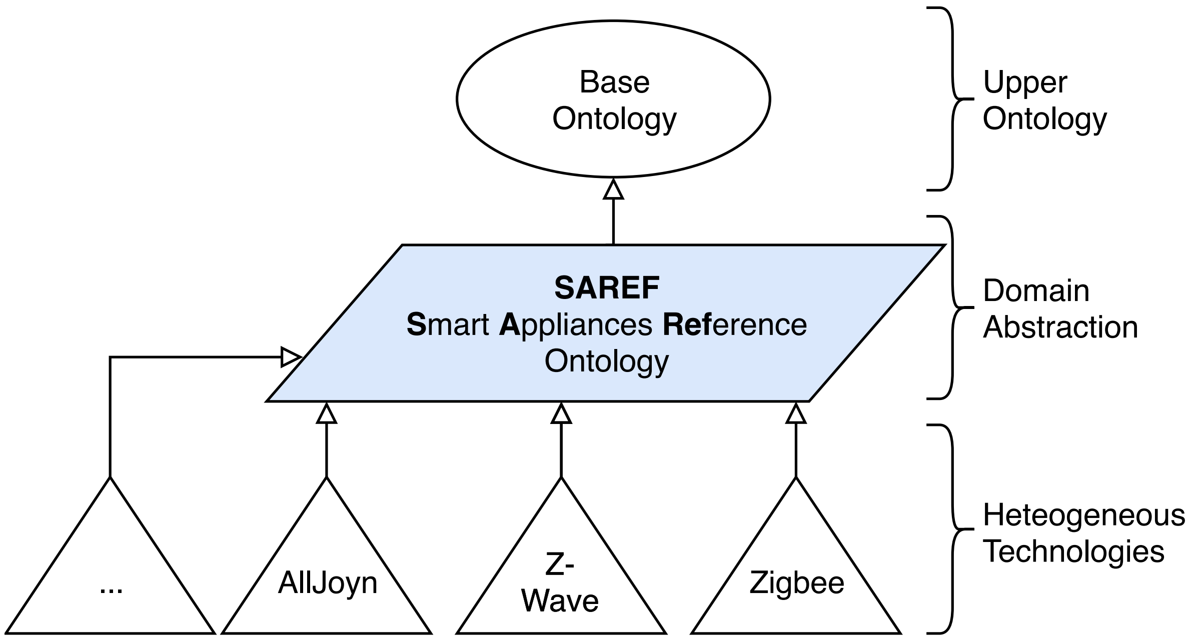


Figure 3.4: SAREF Ontology Context

3.3.2 Ontology Based Data Model

Ontology was introduced to computer science in order to represent real-life concepts that are understandable by computers[52]. The ontology defines an information model to share a common understanding of knowledge and functionalities. The ECHONET Ontology (eOnt) is required to reflect the concept of ECHONET device objects including restrictions and enumerations based on RDF (Resource Description Framework) as shown in Figure. 3.11.

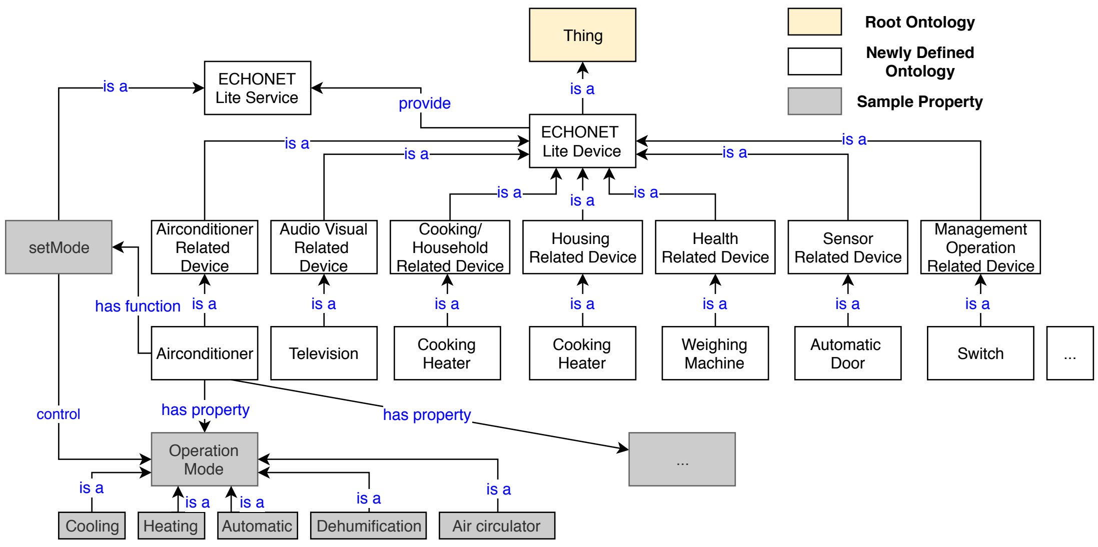


Figure 3.5: Overview of ECHONET Ontology Model

In order to improve the extensibility, the *owl:Thing* is the root of the eOnt which is the most general concept and is easily replaceable by other domain concepts (e.g. *saref:Device*). The structure of the eOnt in Figure. 3.11 is different from ECHONET Lite device concept in Figure. 3.2 where the stating point of the eOnt is the concept of the *device object* instate of *node*, and the *node* information is a part of the *device object*. These changes reduce the *subclassOf* mappings needed and shorten the implementation of a hierarchical mapping for the simple extension with other domains.

The eOnt fully supports 108 types of different devices in the latest ECHONET Lite specifications (English version, Release J, 2017) that includes vocabularies for device properties, relationships and constraints (mandatory or optional, data type and data range restrictions) of properties, and predefined enumeration values.

ECHONET Ontology Metric Analysis

During the study, namely Smart Appliances funded by the European Commission from January 2014 to March 2015, a reference ontology model (SA Ontology) that supports ECHONET Lite specifications were introduced. The SA Ontology exploited ECHONET specification (Release C, 31 May 2013), which supports 89 different device types. The SA Ontology is being used as a reference for evaluation. The quantitative analysis of the eOnt has been calculated by exporting the ontology metric using Protege, a most widely used tool to create and modify ontology [53]. The metric of eOnt and SA Ontology is shown in Table. 3.3

Table 3.3: Ontology Metrics

	SA Ontology	**eOnt** ECHONET Ontology
Axiom	945	**6199**
Classess	188	159
Object Properties	27	**289**
Data Properties	2	**509**
Individual	35	**391**

Axioms support for the semantic interpretation of concepts and relations [54]. More axioms support more inference rules, which can be used for automated reasoning. Numbers of *Data Properties* and *Object Properties* imply the number of properties of a concept that has been mapped to the ontology. The coverage of the ECHONET device objects of the eOnt is better than the referenced ontology model. The number of *Individual* implies the number of mapped enumeration values. The eOnt also supports more individuals to predefined reflex concepts of the specifications. The eOnt model is more comprehensive in the sense of providing semantic interoperability.

3.3.3 Adaptation Layer Architecture

An overview architecture of the AL is described in Figure. 3.6. The AL supports the semantic description of the network of ECHONET Lite devices by mapping device resources and services into RDF format resources using the eOnt model. This AL layer allows integrations between ECHONET Lite protocol with service platforms by implementing corresponding technology exporters, which take the exported semantic resources as input for wrapper functions of specific technologies without any knowledge about the ECHONET Lite interface.

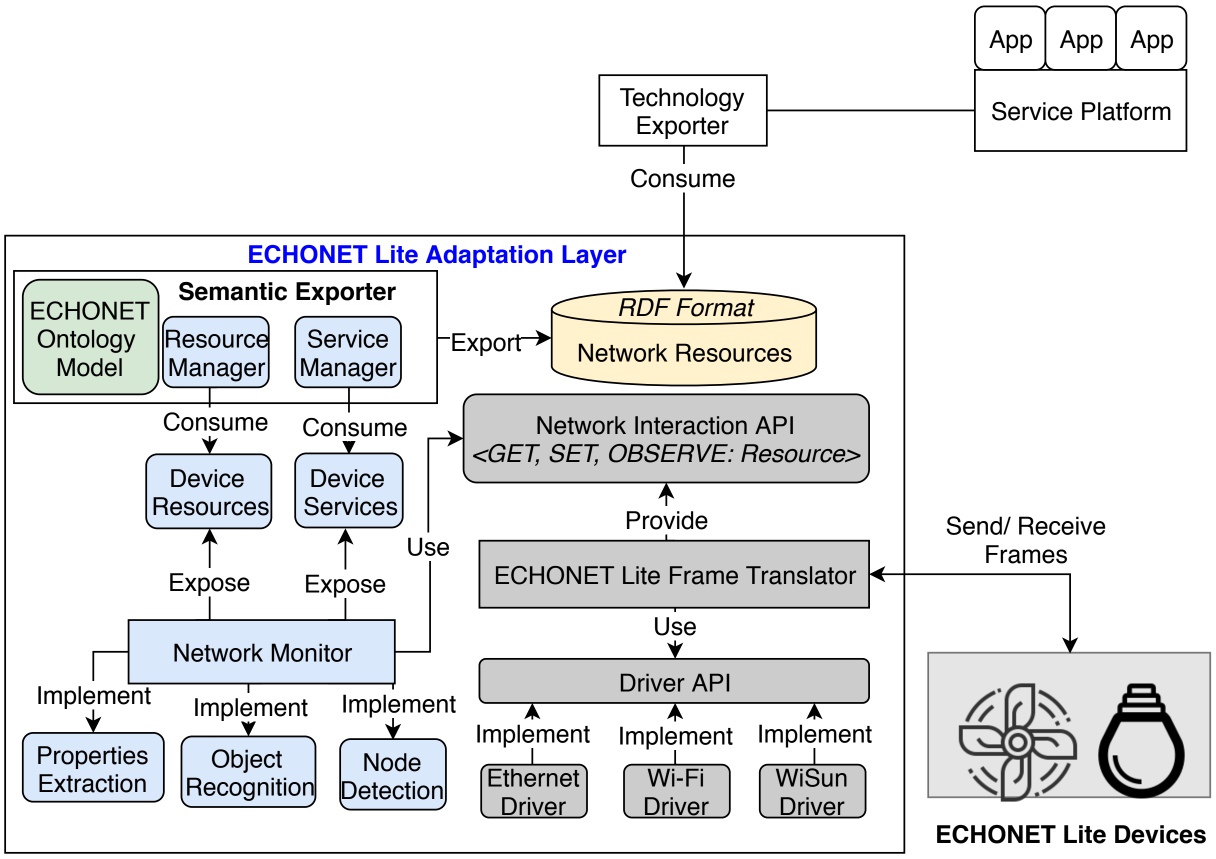
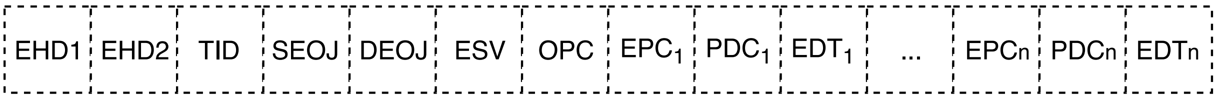


Figure 3.6: ECHONET Lite Adaptation Layer Architecture

Frame Translator

The ECHONET Lite protocol is defined at the application layer of the OSI model where the network layer is based on Internet Protocols, the frame translator (FT) utilizes IP supported drivers such as Ethernet, Wi-Fi, or WiSun driver. The implementation of this FT module is heavily based on the frame format (Figure. 3.7), which is a part of the ECHONET Lite specification.



```
- EHD1: ECHONET Lite message header 1 (1 Byte)
- EHD2: ECHONET Lite message header 2 (1 Byte)
- TID : Transaction ID (2 Bytes)
- SEOJ: Source ECHONET Lite object specification  (3 Bytes)
- DEOJ: Destination ECHONET Lite object specification (3 Bytes)
- ESV : ECHONET Lite service (1 Byte)
- OPC : Number of processing properties (1 Byte)
- EPC : ECHONET Lite Property (1 Byte)
- PDC : Property data counter (1 Byte)
- EDT : Property value data (Specified by PDC)
```

Figure 3.7: ECHONET Lite Frame Format

ECHONET Lite Header 1 is a 1-byte value specifies ECHONET protocol type. EHD1 with value **00010000** indicates ECHONET Lite protocol. **ECHONET Lite Header 2** is a 1-byte value indicates format of EDATA filed. There are two options : **10000010** (EDATA is in arbitrary message format) and **10000001** (EDATA is in Format 1 as describing in Figure. 3.7). **TID** is a 2-byte transaction ID parameter that matches the request and response. **EDATA** is variable-length ECHONET Lite data field of message exchanged between ECHONET Lite devices.

GET, *SET* operations are mapped to the corresponding value of the *ESV* and a *Resource* is described by a triple of *EPC* (property name), *PDC* (property format) and *EDT* (value).

Network Monitor

The main objectives of this network monitor module are to (i) obtain a list of ECHONET Lite nodes in the network, (ii) identify device objects in each node and (iii) recognize supported properties and extract property data in each object.

Node Detection

There are two ways to implement the node detection for the ECHONET Lite network [55] (i) passively waiting for a message sent by nodes when they joined into a network or (ii) actively broadcast a node finding message at an arbitrary timing.

The first approach seems to support the real-time detection of the node without flooding the network with the node finding message. However, this approach is not feasible to apply in an entire running network because devices joined the network, and they will not send the identification message again. The second approach solves this problem by actively asking nodes to send the identification message. However, in order to reduce latency for node detection, a high broadcasting frequency is desired, and it increases overhead for the network.

To this end, the hybrid approach, which only broadcasts the node finding message once at the starting time and passively waiting for messages from nodes, is implemented to take advantage of both mentioned approaches. The node finding message is shown in Figure. 3.8 with the *TID* is an arbitrary value as the transaction ID and *SEOJ*'s value is a node profile object(e.g. *0x0EF001*).

EHD1	EHD2	TID	SEOJ	DEOJ	ESV	OPC	EPC
0x01	0x81	-	-	0x0EF001	0x62	1	0xD6

Figure 3.8: Node Finding Message

Object Recognition

As stated in the ECHONET Lite specifications, a node must return a response with a list of device objects supported by that node. A device object has its identification, which provides the device name. By the device name, supported properties, as well as property constraints of that device, are defined at the specifications. However, in the specification, there are optional properties, and these optional properties need to be identified by the *Property Extraction* module.

Properties Extraction

ECHONET Lite specifications require each device object to maintain a list of settable properties (*EPC 0x9E*), a list of gettable properties (*EPC 0x9F*), and a list of observable properties *EPC 0x9D*. Therefore device services and resources are extractable by sending frames with the corresponding *EPC*s to request the list of supported properties.

Semantic Exporter

This module uses the eOnt data model and device resources to serialize semantic resources in the RDF format, which is a standard for the semantical description of resources. An RDF statement is a triple of a subject, a predicate, and an object. A subject is a resource, a predicate implies the relation between a subject and an object, and an object can be a resource or a data value. RDF statements can be linked to each other by using resources as objects. These resources are the object of one statement, then become the subject of another statement.

The *Service Manager* is responsible for creating resource described generic services for a class of devices which shares the same functionalities. The *Resource Manager* is responsible for creating device resources as an RDF graph. The*ECHONET Ontology Model* provides vocabularies that are used to annotate device resources and services as RDF data semantically.

3.3.4 ECHONET Lite Adaptation Flowchart

The AL monitors device properties data changed events and maps these changes into semantic resources as illustrated in Figure. 3.9.

In the ECHONET Lite specification, observable properties will notify the observer whenever a data-changed event happened. Thus these observable properties are easily monitored by implementing corresponding observers. For other properties, the AL has to store the old value and send multiple unicast messages to query data at an arbitrary timing.

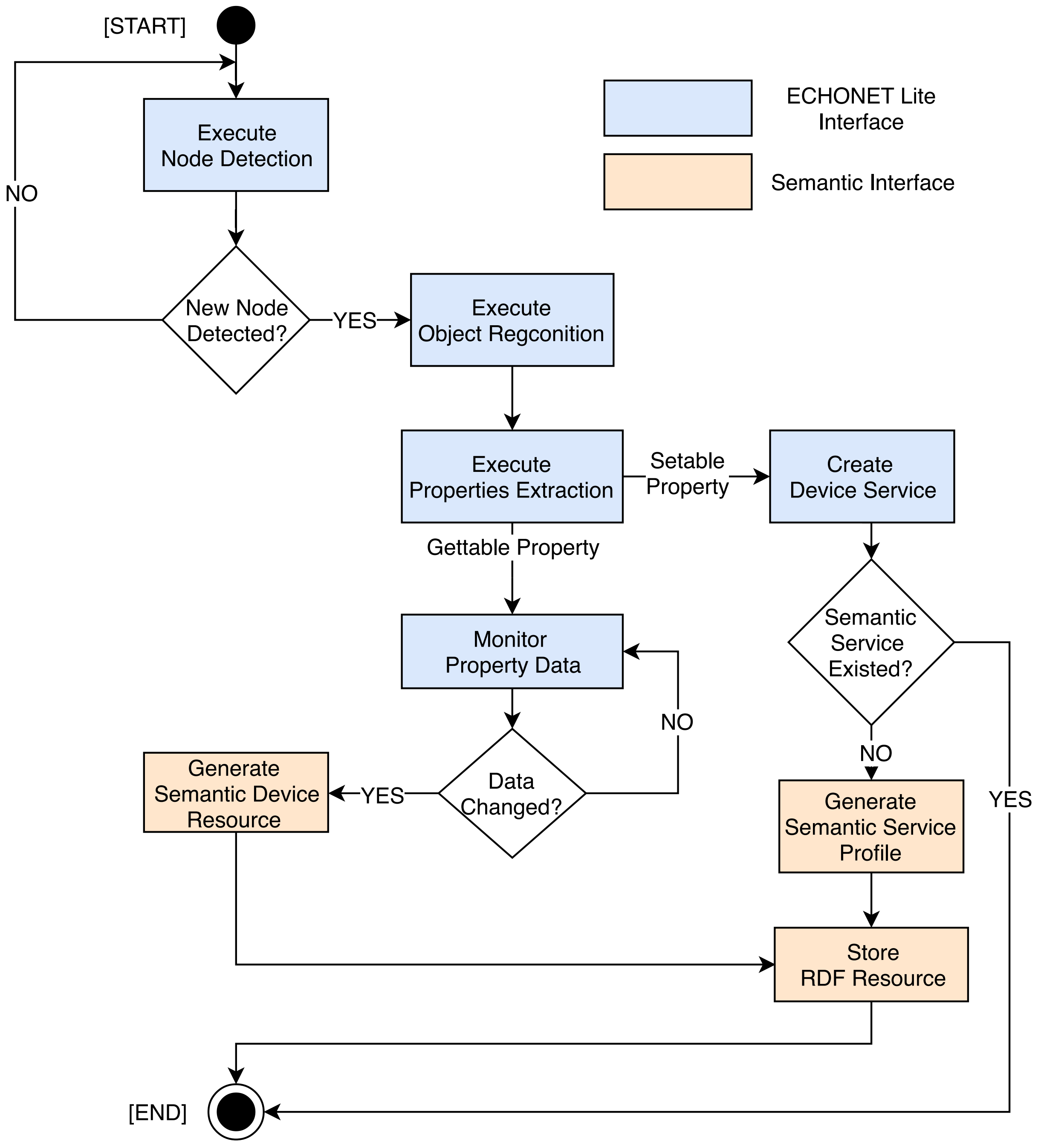


Figure 3.9: ECHONET Lite Adaptation Layer Flowchart

3.4 Integration Layer: oneM2M Integration

The ECHONET Lite protocol lacks interoperability with service platforms and the support of standard APIs for service developers as well. This integration introduces a solution to handle ECHONET Lite device resources and transform these resources into standard APIs usable by third-party applications. An oneM2M Interworking Proxy Entity for ECHONET Lite protocol was implemented, and ECHONET Lite devices are able to have interacted with devices from different domains via the oneM2M ecosystem.

3.4.1 oneM2M and oneM2M Interworking Proxy Entity

oneM2M is a global standards initiative that aims to define a standard service platform for M2M systems. For the interoperability with non-oneM2M systems, the concept of the interworking proxy entity (IPE) (illustrated in Figure. 3.10) was proposed by oneM2M.

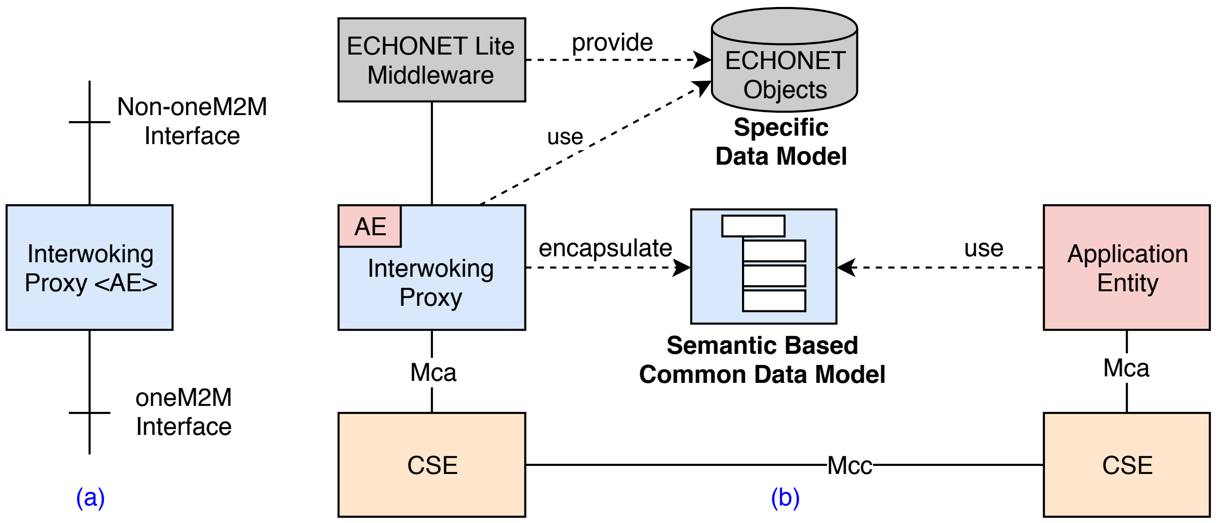


Figure 3.10: oneM2M IPE(a) and its example(b)

The IPE is an application entity (AE) that provides the application logic to map specific data models into a common data model usable by the oneM2M system. A common service entity (CSE) comprises a set of services usable by applications and other CSEs such as registration, security, application, service, data, and device management. The *Mca* and *Mcc* are reference points that enable AEs to use services provided by CSE and inter-CSE communication, respectively.

3.4.2 oneM2M Based ECHONET Ontology

An ontology[56] defines an information model to share a common understanding of knowledge and functionalities of a domain, and it has been used to achieve interoperability of the oneM2M ecosystem with external systems. The oneM2M base ontology enables the semantic discovery of entities in the oneM2M ecosystem. Additionally, it allows non-oneM2M technologies to interwork with the oneM2M system by extending the base ontology concepts. Since ontology supports both syntactic and semantic interoperability[57], it has been chosen as a common data model to represent ECHONET Lite device resources. In **Section.** 3.3.2, a comprehensive ECHONET ontology model was introduced. The proposed ontology model fully covers the ECHONET Lite concept as shown in Figure.**??**. The oneM2M based ECHONET Ontology namely *oneECHONET* is proposing by replacing the root ontology of the domain ontology by the root ontology of the oneM2M based ontology [5].

By this extension, the *oneEchonet* ontology model is compatible with oneM2M ontologies and fully supports the concept of ECHONET Lite device objects. Therefore,

[5]https://git.onem2m.org/MAS/BaseOntology

third-party developers can access to ECHONET Lite resources without prior knowledge of ECHONET Lite via the oneM2M defined concepts.

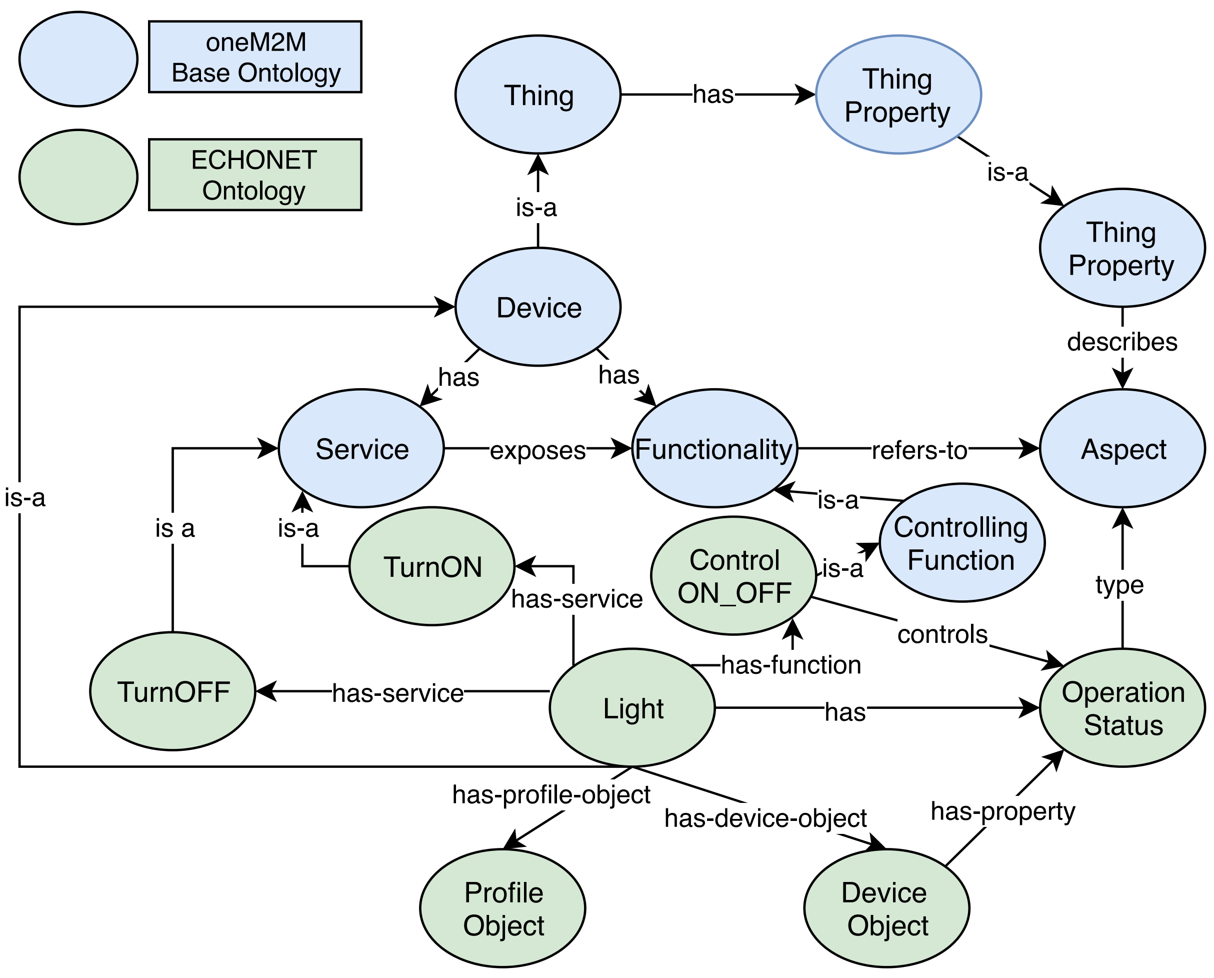


Figure 3.11: Ontology Based Data Model for oneM2M Integration

3.4.3 Integration Layer Architecture: ECHONET Lite Interworking Proxy Entity

oneM2M is a well-known platform and to build the *Common Platform* as stated in ITU-T Y.2070 recommendation [38], proposing the *Home Gateway* (HGW) which is compatible with the oneM2M ecosystem is a must. Concerning the oneM2M compatibility, it could be fulfilled by the concept of IPE and *oneECHONET* ontology. The HGW is required to implement an operational procedure to manage a network of ECHONET Lite devices then map resources into the semantic description, and those semantic resources will be registered to the oneM2M platform by the CSE (provided by the oneM2M IPE). The overview architecture of eIPE is illustrated in Figure. 3.12.

The core components of the ECHONET Lite IPE are:

- The proposed **Adaptation Layer** with the *oneECHONET* ontology model.
- *Resource Manager* maps resources extracted from the *Network Monitor* module, which includes the device name, and a list of *property: data* tuple into semantic

resources by utilizing the oneECHONET ontology model. Then, these semantic resources will be passed to a CSE so that the CSE will register them to the oneM2M ecosystem.

- *Service Manager* acts the same to the *Resource Manager*. However, it only handles *settable properties*, and these registered semantic resources will be used to advertise provided services of an ECHONET Lite device. It also handles requests from CSE by triggering exposed device services.

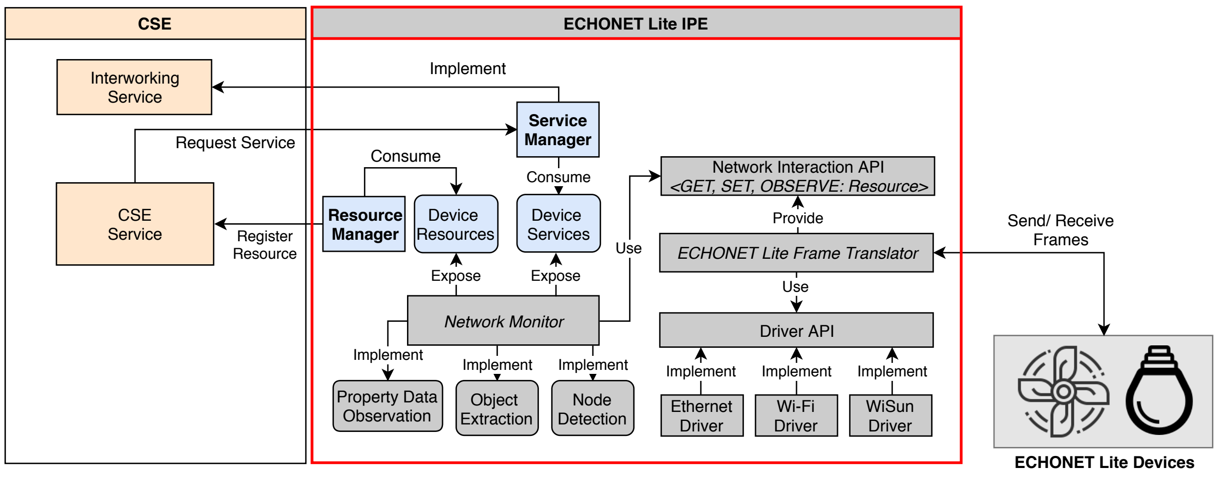


Figure 3.12: Overview of ECHONET Lite IPE

3.4.4 Implementation

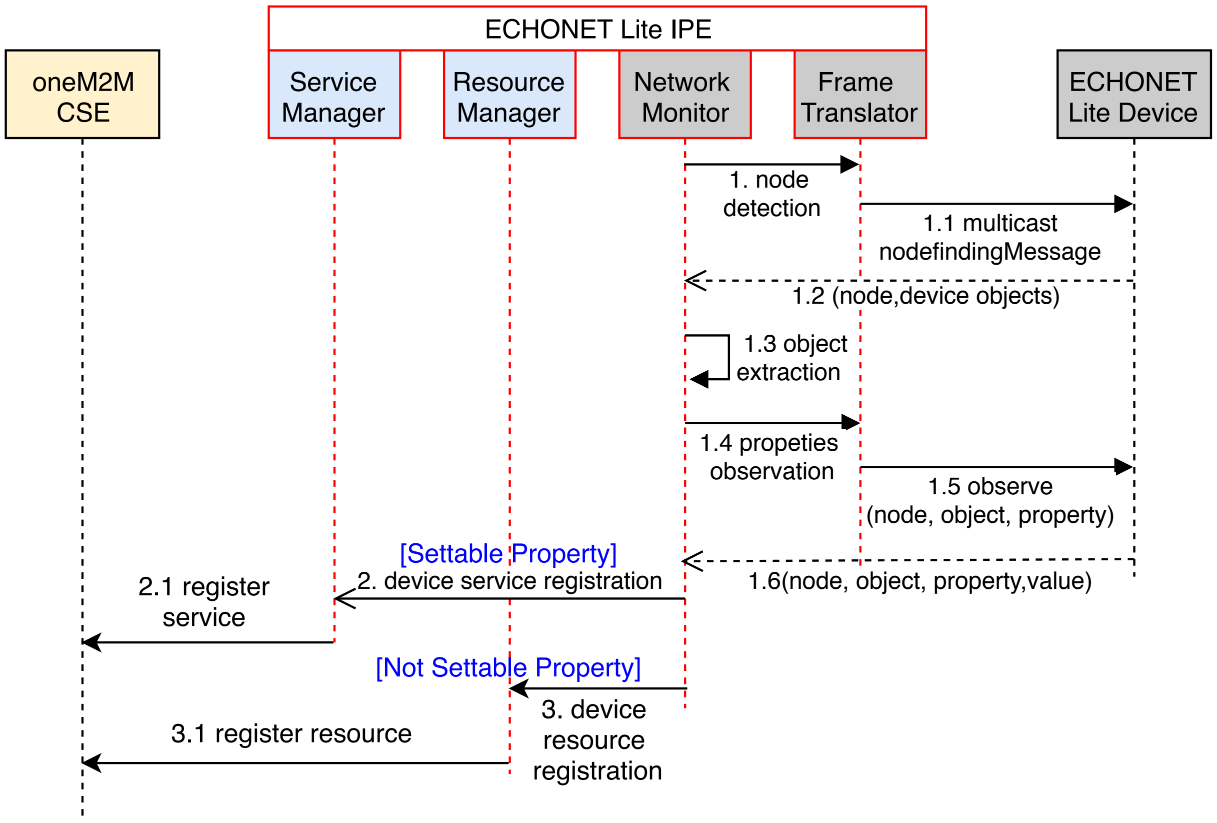


Figure 3.13: Device Resource and Service Registration Sequence

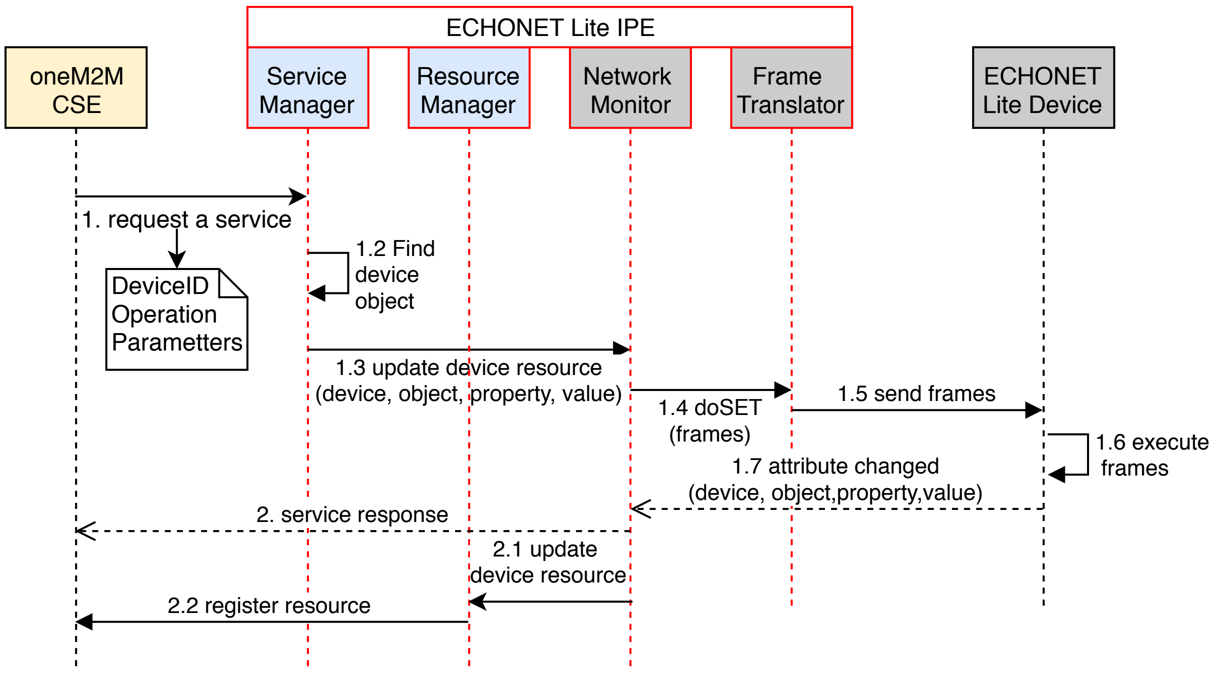


Figure 3.14: Device Interaction Sequence

oM2M[58] is a java based open-source implementation of the oneM2M. The proposed eIPE was implemented as an OSGi bundle and integrated into the oM2M project as the HGW. The sequence diagrams for ECHONET Lite device resources and services detection and registration, as well as device interaction (control) by the proposed ECHONET Lite IPE is illustrated in Fig. 3.13 and Fig. 3.14 respectively.

3.4.5 Demonstration

Apparently, ECHONET Lite devices and a device developed by Amazon, namely Amazon Echo Plus, are incompatible. The purpose of this demonstration is to verify the interoperability and compatibility between them by using the oneM2M ecosystem and the proposed HGW. Two ECHONET Lite lighting devices created by Toshiba (Toshiba LEDD85021LT1) are subject of the demonstration, and the detail configuration of the demonstration is described in Figure. 3.15.

In the end, two lights are controllable by the Amazon Echo Plus, which is not compatible with the ECHONET Lite protocol by utilizing APIs provided by the implemented IPE via the oneM2M service platform.

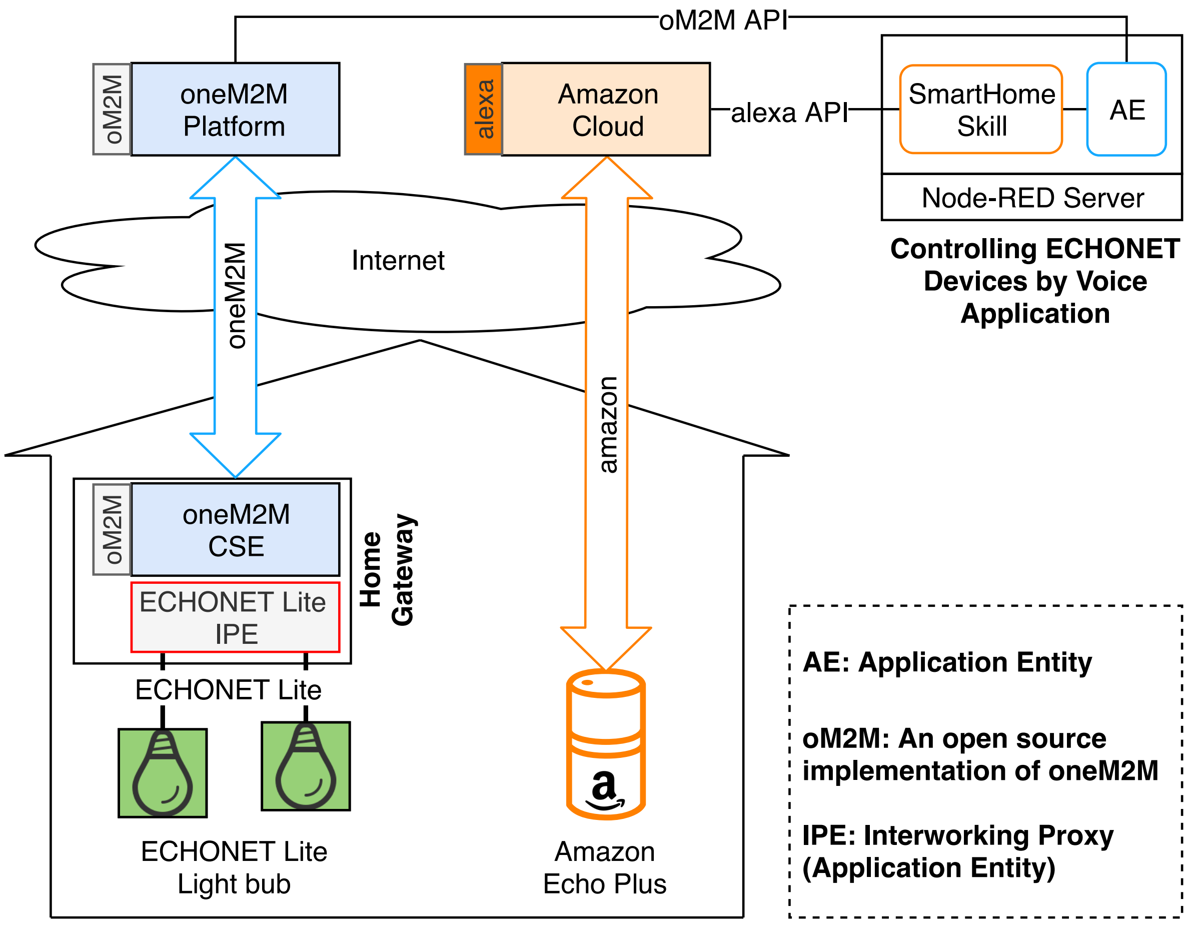


Figure 3.15: Demonstration Configuration

3.4.6 Conclusion

This integration layer introduces the integration of the ECHONET Lite protocol into the oneM2M ecosystem by extending and implementing the ECHONET Lite IPE. The proposed IPE facilitates networks of ECHONET Lite device resources into semantic description compatible with oneM2M and discoverable by third-party developers via standardized API of the oneM2M. Additionally, a comprehensive ontology model called *oneECHONET* is proposed to be a foundation for the semantic interoperability. To verify the feasibility of the proposed solution, a practical implementation of the ECHONET Lite HGW was introduced. The HGW supports seamlessly plug and play as well as allows interaction between ECHONET Lite devices and other appliances and applications which are not in the ECHONET Lite domain.

3.5 Integration Layer: universAAL Integration

3.5.1 universAAL Platform

universAAL (uAAL) [59] stands for universal open platform and reference specification for Ambient Assisted Living and is the result of the European Union funded project to produce an open platform for the AAL. The uAAL platform allows the seamless integration of heterogeneous devices within a network environment through two base concepts: i) the usage of three communication buses for topic-based communication among components, namely a Context Bus, a Service Bus, and a User Interface Bus; ii) the usage of ontologies for information and services sharing between components semantically. An overview of the uAAL platform is shown in Figure. 3.16. uAAL MW is the core component of the uAAL platform, which encompasses the communication infrastructure of the platform. All devices that run this MW are nodes that can share knowledge and functionalities with other nodes in the form of ontology. The heart of this MW is formed by three buses, and all the communication takes place via one of three following buses:

1. Context Bus (CB) is an event-based communication channel to allow nodes to publish context events to the CB, regardless of the existence of recipients or not. Recipients are context subscribers who register their interest to the CB in order to only receive registered events.

2. Service Bus (SB) is a call-based communication channel that allows nodes to request services from other nodes. Service providers are called service callees. They announce themselves by registering a service profile that describes their capabilities to the SB. The counterpart to service callees is service callers, which send a service request through the SB to ask for a specific request.

3. User Interface Bus (UI Bus) is used for delivering messages related to user interactions.

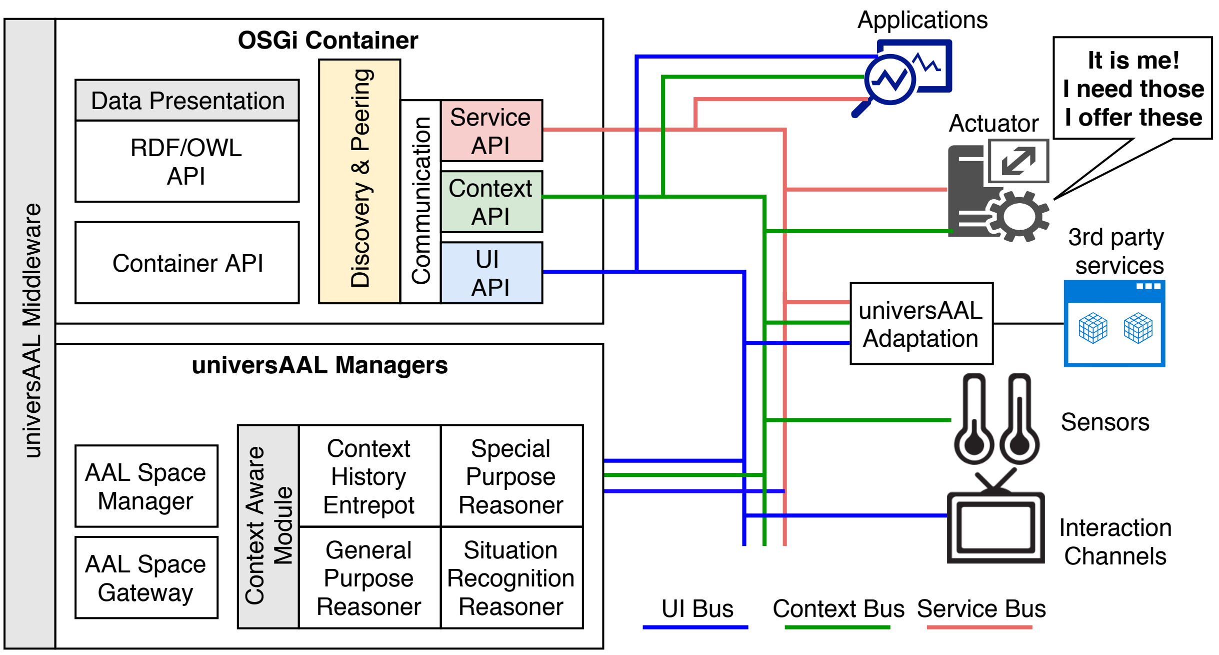


Figure 3.16: The universAAL Platform

The uAAL is the most promising and holistic platform [60], which directly benefits the end-user by being an affordable, simply configured, and personalized solution that also further empowers service providers by enabling easier and cheaper development of new AAL services or adaptation of existing ones. The uAAL has been widely used in European AAL projects such as inLife [61], ACCRA [62], Plan4Act, Plan4Act, OCARIOT, ReAAL, and Activage [63]. The idea of ECHONET and uAAL integration has been proposed in recent researches in Japan. In [64], the basic concepts and benefits of the integration were clarified, in [65], the integration layer, including the ontology model, has been clarified. However, detailed implementation, as well as the integration methodology, have not been explained. This integration is not only to validate the semantic interoperability of the proposed solution but also an effort to make the ECHONET Lite protocol available to European countries as well as support AAL services for ECHONET Lite smart home in Japan.

3.5.2 Implementation

There are two main approaches to protocol-platform integration: *Adapter* and *Gateway* approach. However, there are a considerable amount of heterogeneous devices in smart home environments, and the adapter approach requires one adapter for each device in which the total hardware cost is extremely high. The ECHONET Lite - uAAL gateway (*elite4u*) is implemented for the integration.

The overall architecture of the *elite4u*, which used the proposed AL to facilitate a network of ECHONET Lite devices, is illustrated in Figure. 3.17.

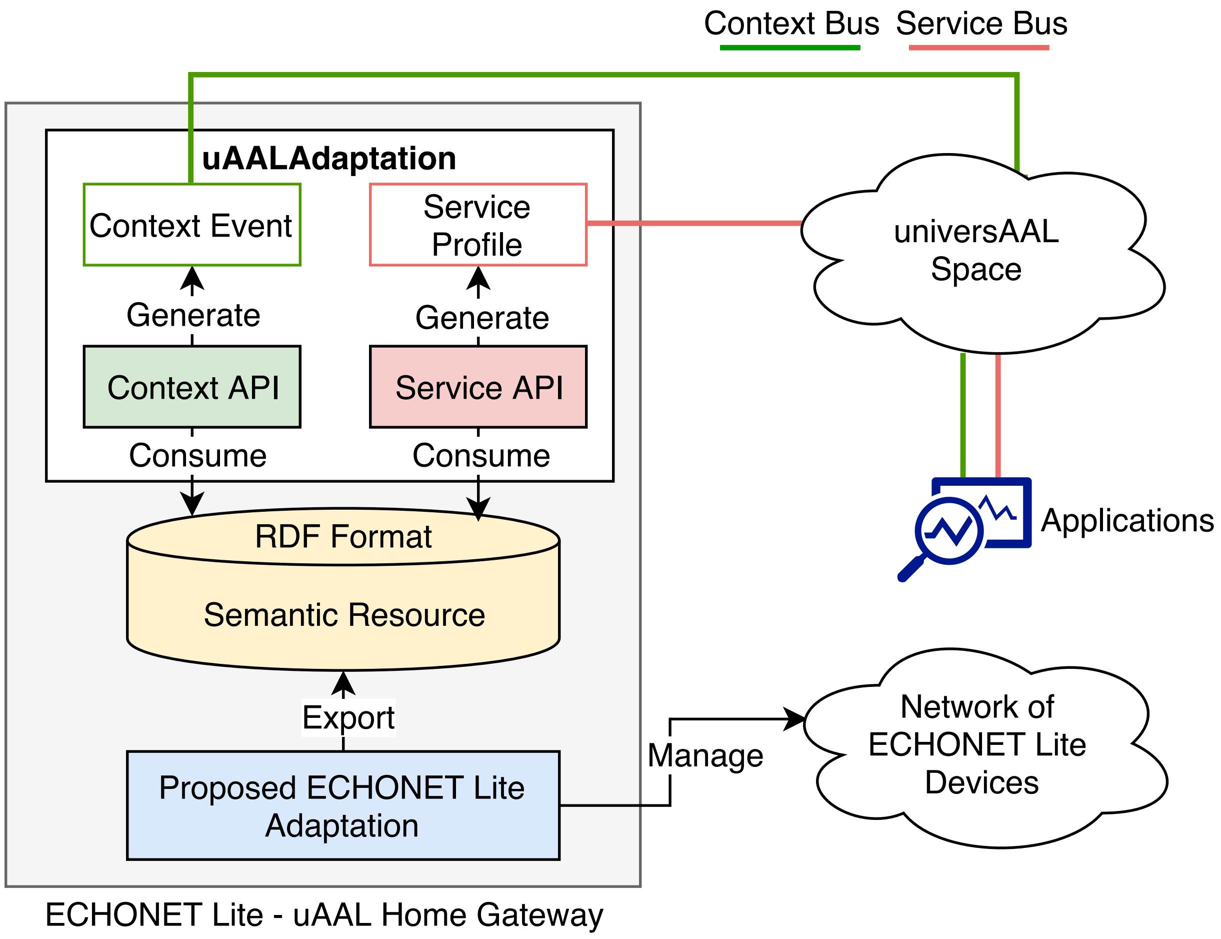


Figure 3.17: ECHONET Lite - uAAL Gateway

The elite4u has been implemented with the following characteristics:

- Seamless Plug and Play: When deploying the elite4u gateway into a network of ECHONET Lite devices, running nodes and newly joined nodes are discoverable. Resources and functionalities of discovered nodes are extractable and manageable. All devices are recognized and integrated automatically.
- Semantic Annotation of resources of the network of ECHONET Lite devices: Discovered logical device properties are mappable into semantic resources, which are discoverable and understandable for the semantic level interoperability support.
- Ontology support: Ontology provides consistent meanings and relationships to describe resources, which is the foundation for semantic annotation. The ontology must be comprehensive in order to reflect all attributes of a device without knowing device specifications.

Since the ECHONET Lite protocol does not provide facilities for UIs, the UI bus was ignored in this architecture. The elite4u exploits semantic resources of the network provided by the AL, classifies resources into knowledge into *Context Event*, and functionalities into *Service Profile* to share in the uAAL space.

3.5.3 Experiment

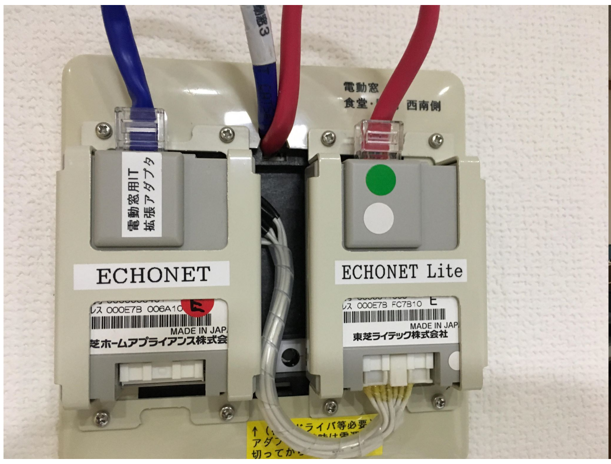
Smart Home Testbed: iHouse

Experiments that have been conducted to test the implemented elite4u in the smart home supported the ECHONET Lite protocol. The experiment has been conducted at the iHouse located at Nomi city, Ishikawa Prefecture, Japan. The outdoor and the living room (experiment room) of the *iHouse* is shown in Figure. 3.18 and Figure. 3.19.

Figure 3.18: iHouse Outdoor

(a) ECHONET Lite Lighting Device

(b) Peper Robot with Controller



(c) ECHONET and ECHONET Lite Adapter

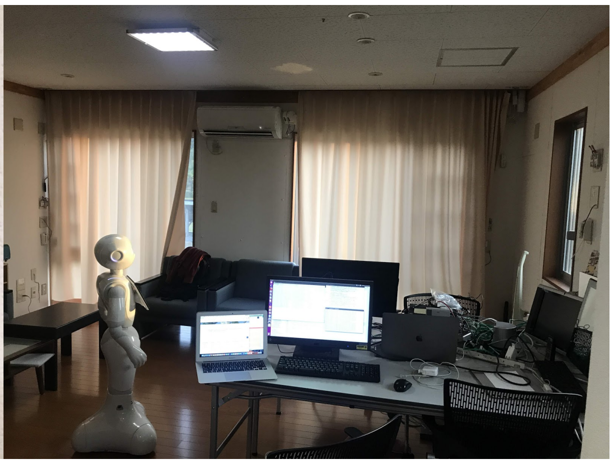

(d) iHouse Living Room

Figure 3.19: Experiment Environment: iHouse and experiment devices

The iHouse is an advanced experimental environment for future smart homes in Japan, and it has been implemented according to Standard House Design by the Architectural Institute of Japan. The iHouse consists of sensors, electronic devices, and home appliances that are connecting by utilizing ECHONET Lite version 1.1 and ECHONET version 3.6. This configuration network emanates more than 300 sensors and actuators.

Experiment Configuration

All the tests have been carried out at the iHouse with the configuration as in Figure. 3.20

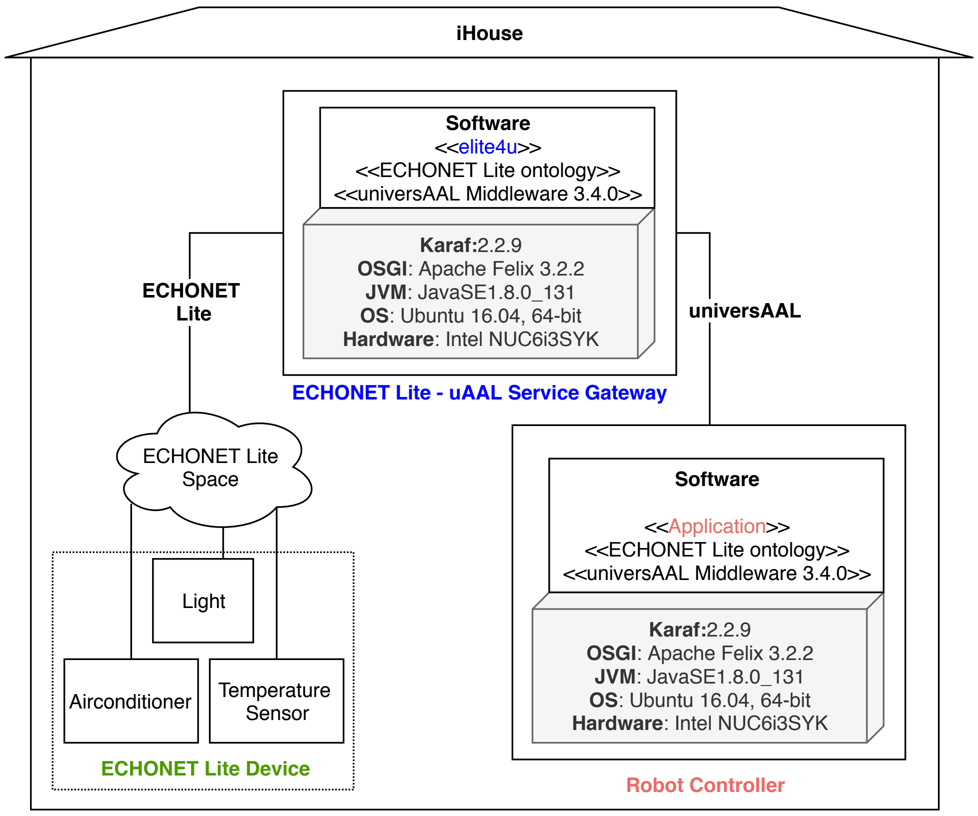


Figure 3.20: Experiment Configuration

The elite4u service gateway (GW) is deployed in the Karaf container, which was configured as an uAAL coordinator node. The proposed ECHONET ontology and uAAL MW (v3.4.0) were installed in this Karaf container. The GW has two interfaces in order to interact with the ECHONET Lite network and uAAL space. The Robot Controller (RC) acts as a uAAL node with the same hardware and is configured as a normal uAAL node.

For the experiment, the identification *temperature sensors*, *air conditioners*, and *lighting devices* are recognized fro the interaction, other devices are detectable without identification. There are two use cases (UC) have been conducted. The actor of these use cases includes the *elite4u* service gateway, the entire network of ECHONET Lite device in the iHouse, and the RC.

Use case 1: Seamless Plug and Play

The goal of UC 1 is to verify the operation of the elite4u when installing in the iHouse where the entire network of ECHONET Lite devices is operating normally. The purpose of this UC is to accomplish the following objectives:

- The elite4u GW is able to detect ECHONET Lite nodes and objects in the iHouse;
- The elite4u GW is able to extract and monitor properties of identifiable objects;

- Extracted properties are able to be transformed into semantic resources (context events and service profiles);

The sequence diagram of the UC1 is shown in Figure. 3.21.

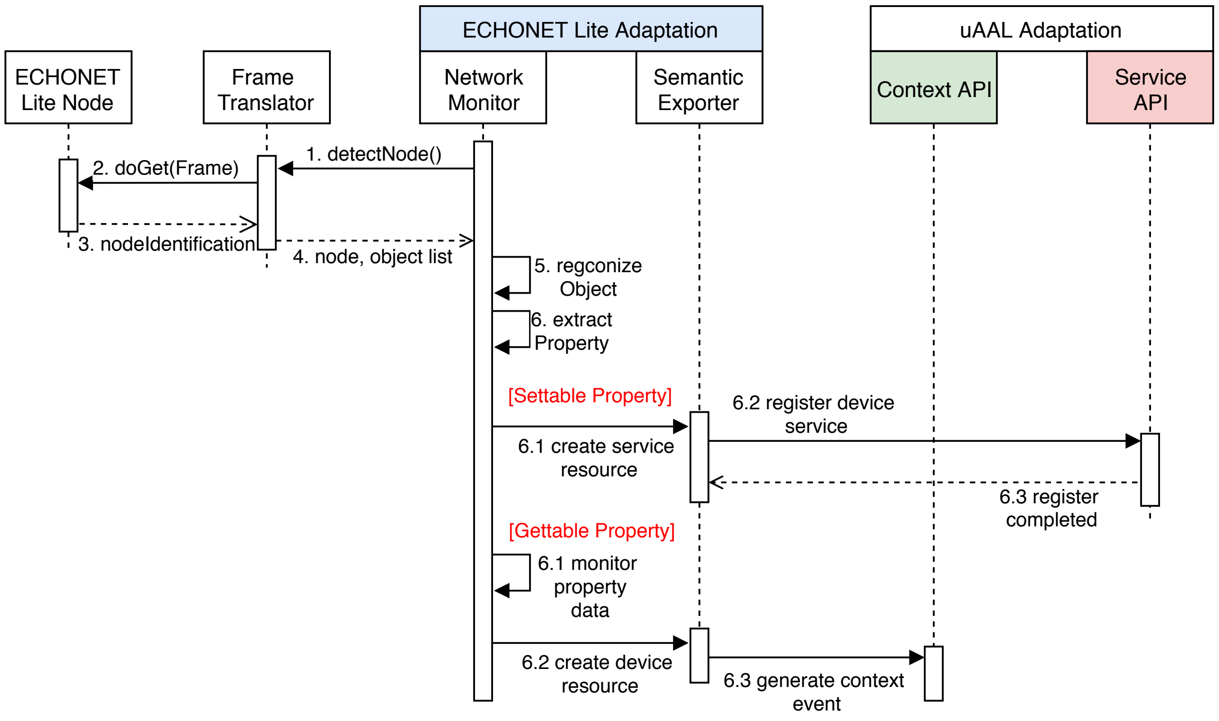


Figure 3.21: Seamless Plug and Play Use Case

As soon as getting started, the elite4u queries connecting nodes of the network by sending node finding message to the multicast address at *224.0.23.0* for IPv4 or *ff02::1* for IPv6. Nodes must return a node identification message which contains a node profile and a list of objects managed by that node. Each object is a logical device, and the name of the object can be identified by *Group Code* and *Class Code*, as stated in the ECHONET Lite specifications. Attributes of the device are extracted as object properties where settable properties allow the interaction will be mapped to services, and gettable properties are resources that must be monitored in order to get the latest status of the device. Device resources and services are semantically annotated by context events and service profiles of the uAAL, respectively.

In order to verify the plug and play, numbers of ECHONET Lite node, numbers of detected object, and the necessary time for this process have been summarized in Table 3.4.

Table 3.4: Node Detection Result

	Total number	Total time for detection(ms)	Average time for detection (ms)
Node	85	564	~6.6
Device Object	252	581	~2.3

Total time to detect network resources is **581 ms**.

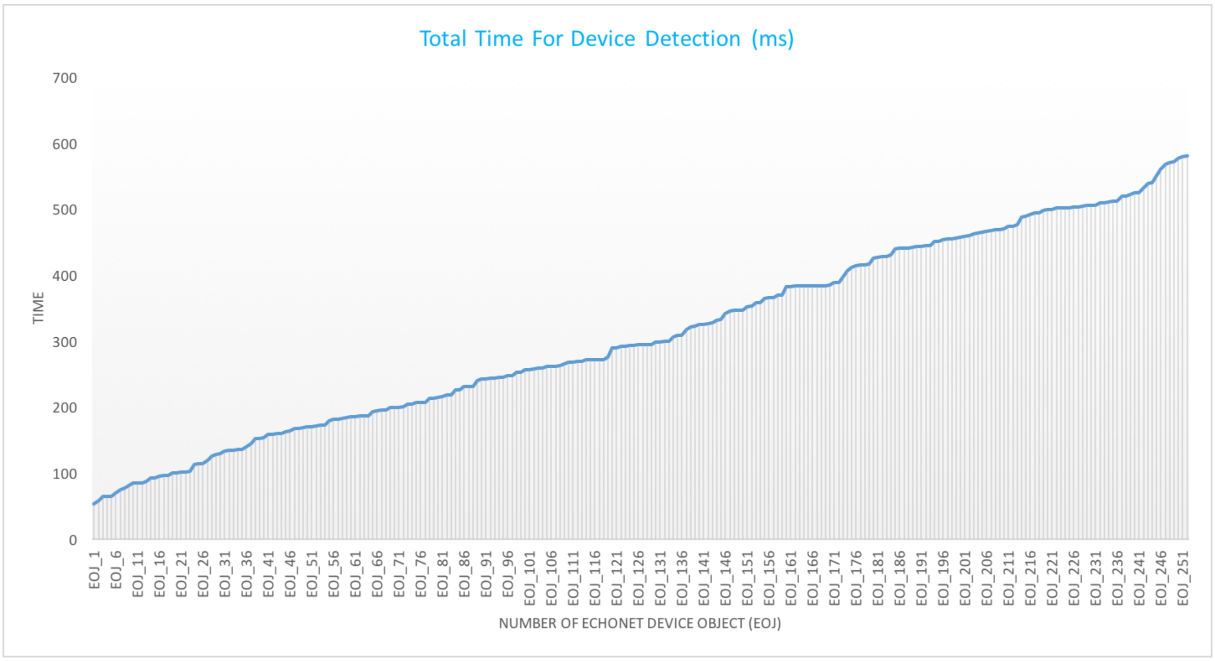


Figure 3.22: Total Time For Device Recognition

Use Case 2: Device Service Interaction

The goal of UC2 is to verify the correct operation of elite4u service gateway when applying commands to devices by the following objectives

- The elite4u is able to register semantic device services exported by the proposed adaptation layer to the service bus.
- The registered services are usable for interacting with devices.
- The elite4u is able to send commands to the corresponding device on the network.
- The elite4u is able to report results sent by the device to the requester.

Recalling the procedures of the elite4u when detecting a node, device services are identified by the access rule of properties. A mechanism to register abstract service profiles for devices that share the same function instead of registering separate service profiles for each device to reduce workload for the service bus was provided by the adaptation layer. The purpose of this use case is to measure the delay time caused by the adaptation layer, including the time to identify the correct device to execute the request.

The sequence diagram of the UC2 is shown in Figure. 3.23.

The uAAL service bus supports a matchmaking mechanism to ensure the semantic discovery of provided services based on pre-registered service profiles, and the delivery of service requests to the service provider is guaranteed by the service bus (1).

As soon as a request is passed to the uAAL adaptation of the elite4u service gateway, the service request is forwarded to the semantic exporter (2) in order to extract corresponding endpoint(3), operation, and parameters (4). After deciding the command, the corresponding ECHONET Lite device service provided by the network monitor module is invoked (5), and ECHONET Lite frames are generated (6). The *SET* operation API provided by the frame translator is called to forward the frame to the corresponding hardware (7). The ECHONET Lite device, which receives the request (in the form of ECHONET Lite frame), executes the request and returns the result frames (8). Based on the frame, a service execution result and a service response are generated accordingly(9), (10), (11). The service bus forwards the service response to the requester and terminates the interaction cycle (12).

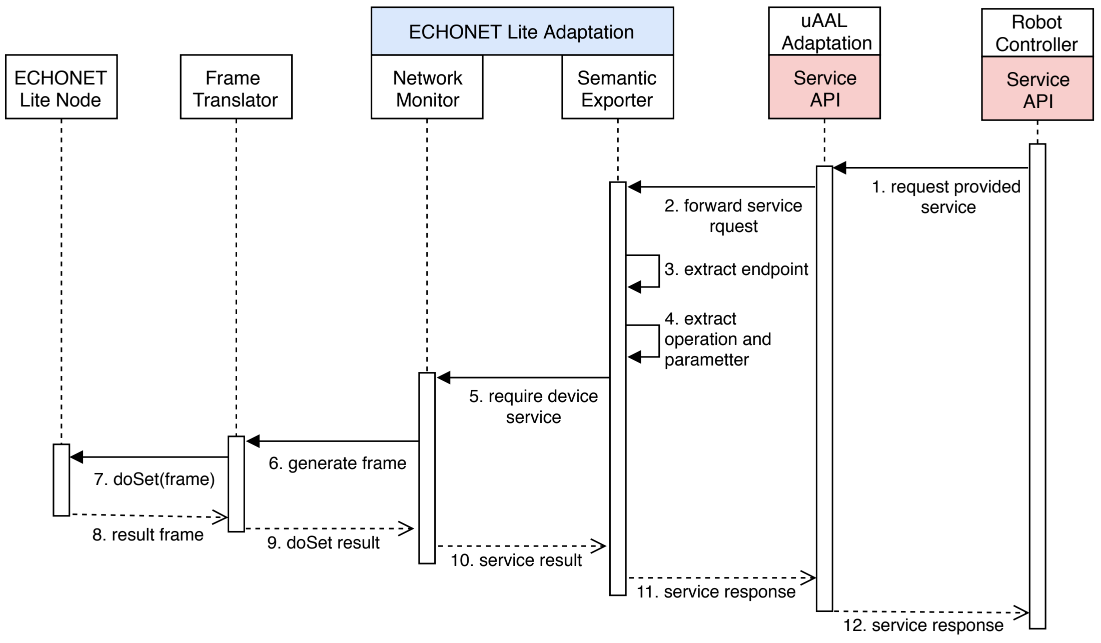


Figure 3.23: Device Service Interaction Use Case

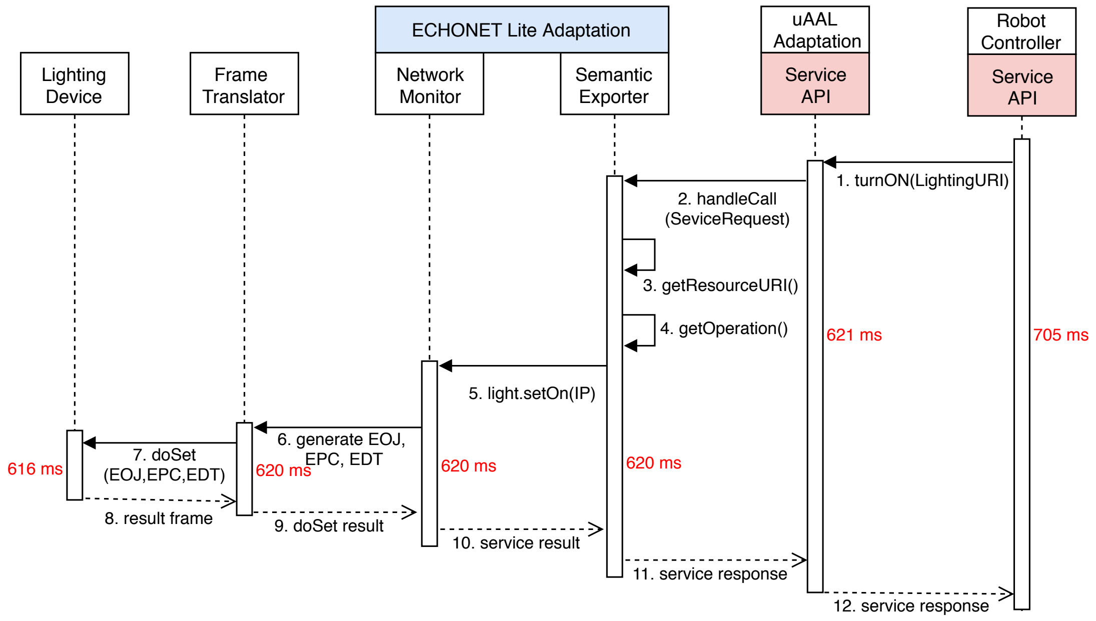


Figure 3.24: Lighting Service Interaction Time

The experiment which used the Robot Controller to turn on the ECHONET Lite lighting device (*Toshiba LEDD85021-LT1*) was conducted, and the time is broken down in

Figure. 3.24.

The necessary time for the light to execute the request is **616 ms**, and the total time for an interaction cycle in the uAAL space is **705 ms**. The time added for the integration is **89 ms**, which is consider a small amount of time. The added time caused by the adaptation layer is only **4 ms**.

3.5.4 Conclusion

The service gateway based on the proposed adaptation layer, which integrates the ECHONET Lite protocol into the uAAL platform was implemented. The wrapper for the uAAL platform was created without taking care of ECHONET specifications by using semantic resources exported by the proposed adaptation layer. The delay time caused by the adaptation layer is small enough to prove the feasibility of the proposed solution. The implemented service gateway is the core component to support the interaction between the robot and smart environment in [66] as a part of the CARESSES project [67].

3.6 Summary

Unlike the *core network*, that is operated by ISP companies with expert network operators, the *IoT area network* (Figure. 3.1) is still required efforts to provide OAM services as users are normal people without knowledge on networking. Since the ECHONET Lite protocol covers all requirements of desired *IoT area network* protocols stated in [42], the ECHONET Lite protocol has been chosen as target protocol for the home network model. To implementing AI models for providing OAM services for the *IoT area network*, traffic data of a target home network is desired. As the required traffic data is generated based on the communication between HGW and devices, this section proposed, implemented, and verified an ECHONET Lite based HGW architectures. The proposed layered architecture of the ECHONET Lite HGW supported all requirements of the HGW, as stated in 2.5, and an experiment involves real users was conducted as well. The efficiency and reliability of the proposed HGW were confirmed. The implemented HGW is using as the service gateway for the human, robot, and smart home interaction of the CARESSES project[6].

[6]http://caressesrobot.org/en/

Chapter 4

Device Emulator

4.1 Device Simulation

Since the ECHONET Lite protocol is able to support various protocols/ technologies usable by either constrained and non-constrained devices, the device simulation is proposed and implemented based on the ECHONET Lite protocol.

Table 4.1: Requirements for Device (referred from ITU-T Y.2070)

Requirements	**Functions**	**Description**
Device Operation	Device Object	To support an abstract data model representing resources and functionalities of the device
Management	Managed Agent	To respond to resource information collector request sent from home gateway. Moreover, it is required to check the status of device and report in the case of failure

Table 4.1 summaries requirements of a device as stated in the ITU-T Y.2070 [38].

4.1.1 Related Work

ECHONET Lite is the country's recommended protocol for smart homes in Japan. There have been many efforts to promote the protocol by introducing middleware, tools, and emulators. The summary of the ECHONET Lite software development kit (SDK) referred from the HEMS (ECHONET Lite) Interoperability Test Center [1] is shown in Table 4.2.

[1]http://sh-center.org/

Table 4.2: ECHONET Lite SDKs

Name	Description
SSNG SSNG for iPhone SSNG for NodeJS	Tools to send and receive ECHONET Lite packet Support graphic user interface (GUI)
Device Emulator	ECHONET Lite device emulator Display sent and received ECHONET Lite frames Support GUI
EL Lighting EL Blind	Controller for ECHONET Lite devices Mobile application to control device object via GUI
node-echonet-lite	Middleware supports creating, parssing, sending and listening for ECHONET Lite packets in Node.js. It allows creating and managing device objects
OpenECHO	Middleware supports creating, parssing, sending and listening for ECHONET Lite packets in Java. Also, It allows creating and managing device objects

Currently, ECHONET Lite is getting more attention, especially in European countries, by a collaboration project [67]. These tools and emulators are helpful to get started since ECHONET Lite equipment are not popular in Europe. However, it is challenging to build a smart home simulator by using these SDKs because it is not flexible enough to simulate a new device rather than predefined devices.

4.1.2 Device Emulator

The overview architecture of the proposed ECHONET Lite device emulator is shown in Figure. 4.1.

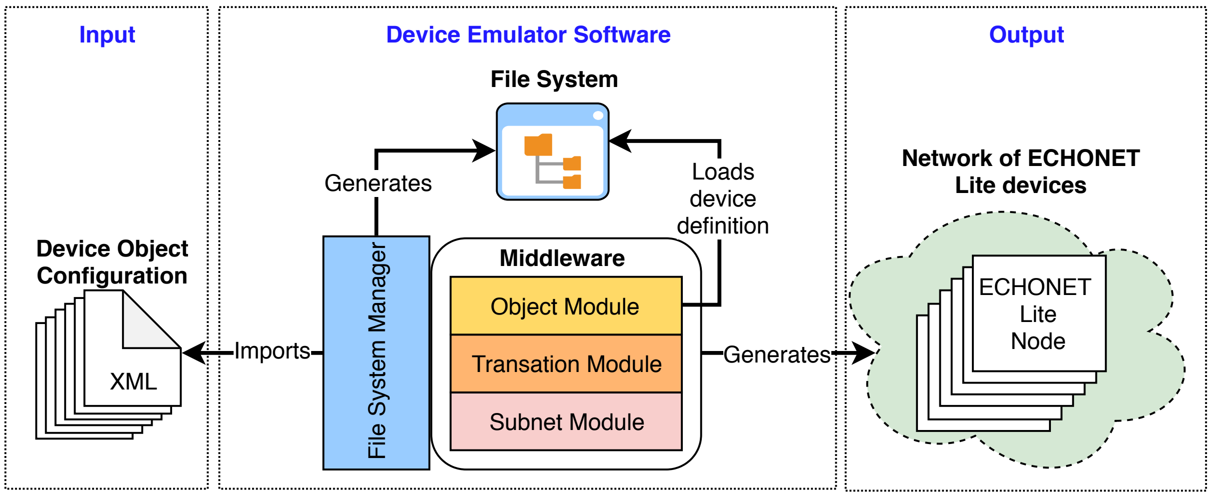


Figure 4.1: ECHONET Lite Device Emulator

The device emulator (DE) is designed while keeping in mind the following points:

- Decoupling the *Device Objects* and *Communication Middleware* (in Figure. 3.2) to improve flexibility when simulating new devices.

- Supporting a mechanism to simulate a faulty device.
- To reduce memory usage, GUI is not necessary. However, an alternative interface for device interaction must be supported.
- Be able to simulate all classes of devices (113 classes in [45]).
- Be able to simulate a network with hundreds of devices.

The purpose of this DE is to take device configuration files as the input and create ECHONET Lite nodes that behave exactly the same to a commercial device by implementing the middleware specification provided by the ECHONET consortium[2]. The proposed DE includes two main components that cover two parts of the ECHONET Lite specifications (Figure. 3.2 a) (i) *Device Objects* and (ii) the *Communication Middleware* and a mechanism to interact with emulated devices via a file system (FS) from outside of the DE.

- Device object configuration (DOC) is an xml[68] document that provides device identification, device resources, and services. The structure of the DOC is visualized in Figure. 4.2 and **Listing** 7.1. The network address is either an IP address or the MAC address of a node and specified by the *file name* of the DOC. The DOC reflects concepts of the *Device Object* in which device resources are properties with initial values and services are specified by *Access Rule*. Each property is attached with a unique *Access Path*, and it is a relative path of the property in the FS. The property value can be updated using this path.

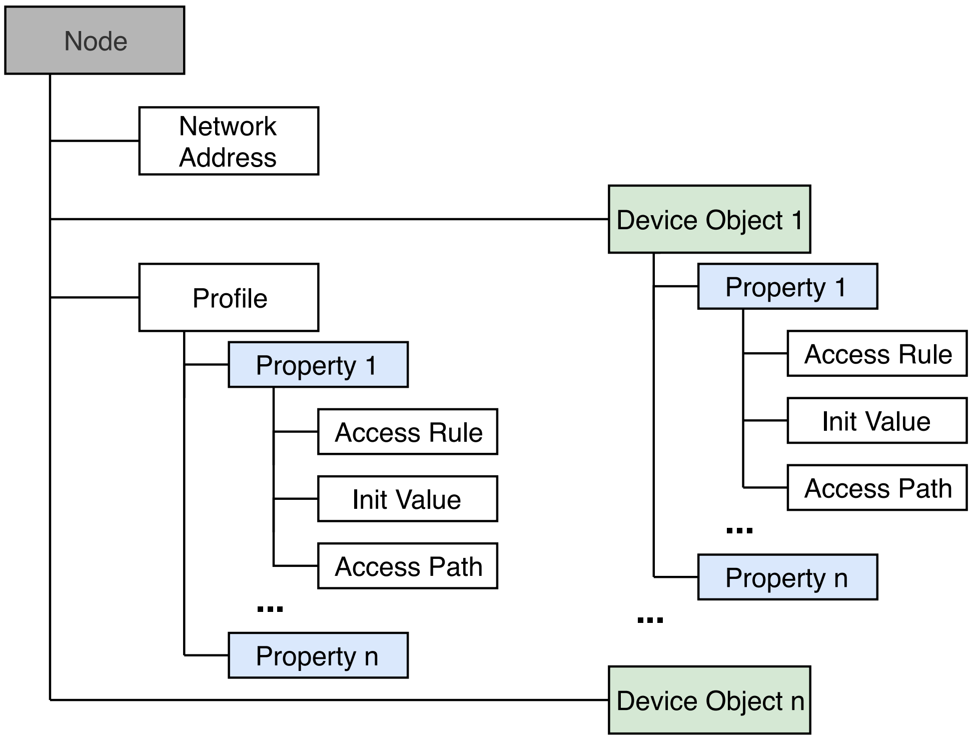


Figure 4.2: Device Object Configuration Structure

[2]https://echonet.jp/english/

- Middleware (MW) represents the standard operating procedure of an ECHONET Lite node, including start-up (restart) sequence, request handling sequence, and notify sequence, which is required by the ECHONET Lite protocol stack. The proposed MW provides three layers (i) *Object Module* imports object definition from the DOC and also monitors the data object value changed event in the FS. (ii) *Transaction Module* manages transactions of request-response cycle and notify cycle. (iii) *Subnet Module* implements network drivers (Wi-Fi, Ethernet, Bluetooth, Wi-SUN, etc.) and a frame translator to translate the ECHONET Lite frame into data and vice versa.

An emulated device can be generated by defining a DOC and importing it to the MW. The FS is respectively generated based on the *Access Path* of each device object. By updating the values of properties via the FS, device status can be simulated by external agents. The pseudo-code of the DE is as in the **Algorithm** 1.

Algorithm 1 ECHONET Lite Device Emulator Pseudo code

```
Input: An XML file
Result: An ECHONET Lite node
Create root folder with the name of XML file
 Create ECHONET Lite profile object
 Create a sub-folder with the name is the profile object code under the root
 for XML tags in XML file do
    for each pair of property, value in the profile object tag do
       Create a file with the name of the property and the content is set with value
        Add the property with default value is value to the profile object
    end
    for device object tag in the XML file do
       Create ECHONET Lite device object
        Create a sub-folder with the name is the device object code under the root
        for each pair of property, value in the device object tag do
           Create a file with the name of the property and the content is set with value
            Add the property with default value is value to the device object
       end
    end
end
```

4.1.3 Faulty Device Simulation

A faulty device could be simulated by mimicking abnormal behaviors (faults) of commercial devices. In [69], [70], [71], and [72], faulty behaviors of IoT devices and wireless sensors based on the data-centric approach are specified. The data-centric approach focuses on data values reported by devices, and a fault can be determined by a data point that deviates from the expected data. These faults can be represented by simulating the value of properties of a target device via the FS without changing the MW or the DOC.

From the view of communication, as stated in [73], a fault can be categorized into:

- **Crash Fault**: A device completely stops responding.

- **Omission Fault**: A device does not reply to one or more requests.

- **Timing**: Responses of a device occur outside of the specified time interval.
- **Response Fault**: A device replies incorrectly either by an incorrect request return value or an incorrect state transition.

These faults required modification of the MW, which handles transmitting and receiving frames. The *crash fault* happened when a device totally drops either the incoming frames or outgoing frames, and it can be implemented by adding a mechanism to drop incoming/outgoing frames to the *Subnet Module* of the MW. The *omission fault* shares the same phenomenon to the *crash fault*, but since it happens with a possibility, the drop rate is less than 100 percent. Therefore, *crash fault* and *omission fault* are simulated by a frame dropping mechanism configurable (drop rate) by a value specified in the DOC. The *timing fault* is simulating by adding a time delay before transmitting frames to the *Subnet Module*. The added delay time is configurable via the DOC also. The *response fault* can be implemented by editing the frame value at the *Subnet Module* to a faulty value before transmitting the outgoing frame.

4.1.4 Implementation

The simulator middleware, namely *humming*, is implemented and released as an open-source project via github[3]. *Humming* is a Java implementation that supports all operations of an ECHONET Lite node. The flowchart of the *humming* is shown in Figure.4.3.

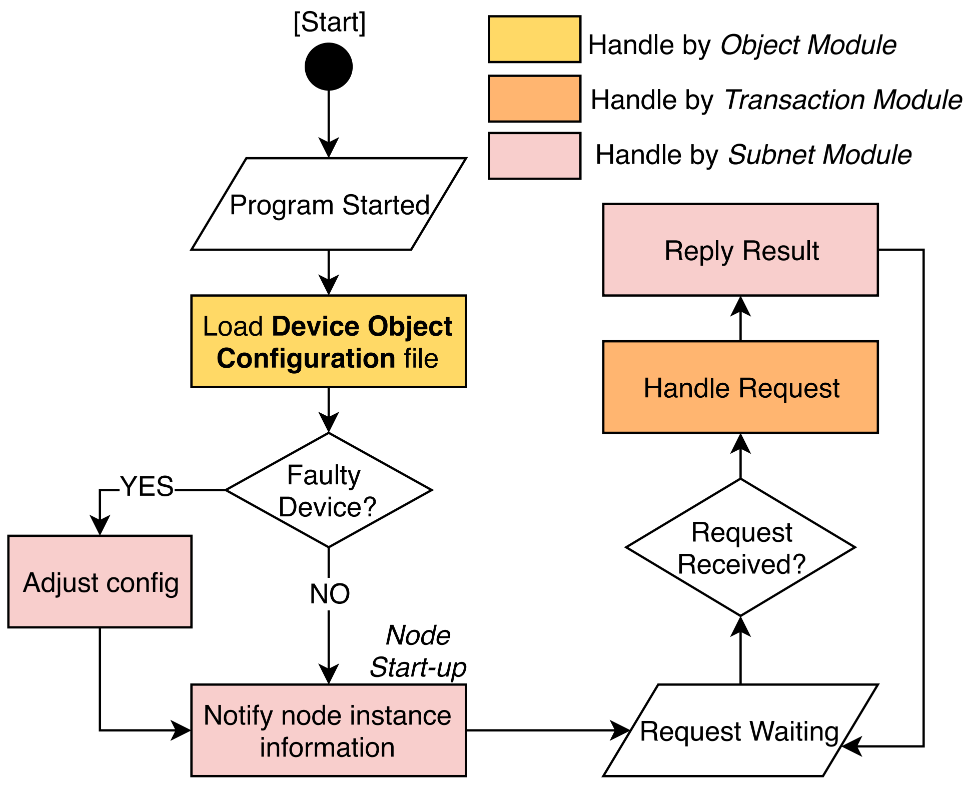

[Start]
Handle by *Object Module*
Handle by *Transaction Module*
Handle by *Subnet Module*
Program Started
Load **Device Object Configuration** file
Faulty Device?
YES
NO
Adjust config
Node Start-up
Notify node instance information
Request Waiting
Request Received?
Handle Request
Reply Result

Figure 4.3: Simulator Middleware (*Humming*) Flowchart

[3]https://github.com/ymakino/humming

A node is created by deploying the MW together with a DOC that describes the node. The *Object Module* loads the DOC file and extracts the node's configuration, which contains information to simulate a target node. The DOC is generated by mapping all required properties of the target node stated in [45] into the XML format. Additionally, the DOC also provides instructions to simulate a faulty node (device). When a faulty node is desired, the configuration is applied to the *Subnet Module* as follows:

- **Crash fault** is falling into two cases (i) device keeps rebooting, or (ii) device does not reply to any request. Since an ECHONET Lite node must notify the *node instance information* which contains node identification and supported device objects at the time the node joins a network, the rebooting scenario is simulating by keep sending the *node instance information* after an arbitrary short timing (about one second) to the multicast address of the network. The non-response scenario is simulating by declining all incoming requests after joining a network, and it is implemented by setting the rate to drop incoming frames to one hundred percent.
- **Omission fault** and **Timing fault** are normally adjusting the rate to drop incoming frames and the delay time before sending outgoing frames at the *Subnet Module*.
- **Response fault** has several patterns, such as replying a request with a wrong result, replying to a request by a non-ECHONET Lite frame, or replying to a request by an invalid value. In this implementation, the response fault is implemented by removing all properties of the target device object from the DOC.

4.1.5 Deployment

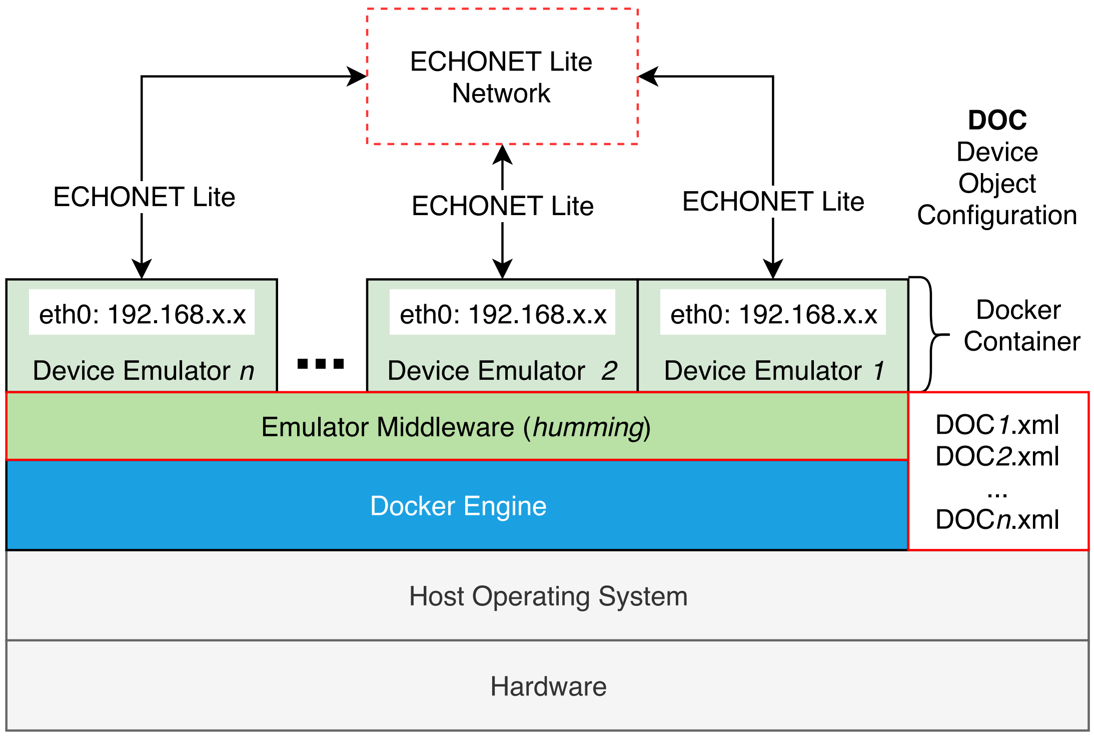


Figure 4.4: Deployment Overview

Algorithm 2 ECHONET Lite Device Emulator Pseudo code

```
Input         : A collection of XML file
Result: ECHONET Lite nodes
Prerequisite: Install docker images and dependencies 4 Install Device Emulator Mid-
              dleware
Initialize a virtual network interface for all containers
for An XML file in the collection of XML file do
    Deploy docker container
    Load emulator configuration information
    Run Device Emulator as in Algorithm 1
end
```

To support an easy and scalable deployment, each DE is deployed as a docker container [74]. The overview of the deployment model and deployment script is as in Figure. 4.4 and Algorithm. 2 respectively. By utilizing the deployment script, the automatic deployment of a large number of ECHONET Lite DE is achievable by feeding the XML file into the target collection only.

4.1.6 Experiment and Evaluation

Normal Device Simulation

Firstly, an experiment to evaluate operational aspects of simulated devices and commercial devices is conducted, and the experiment configuration is as in Figure. 4.5.

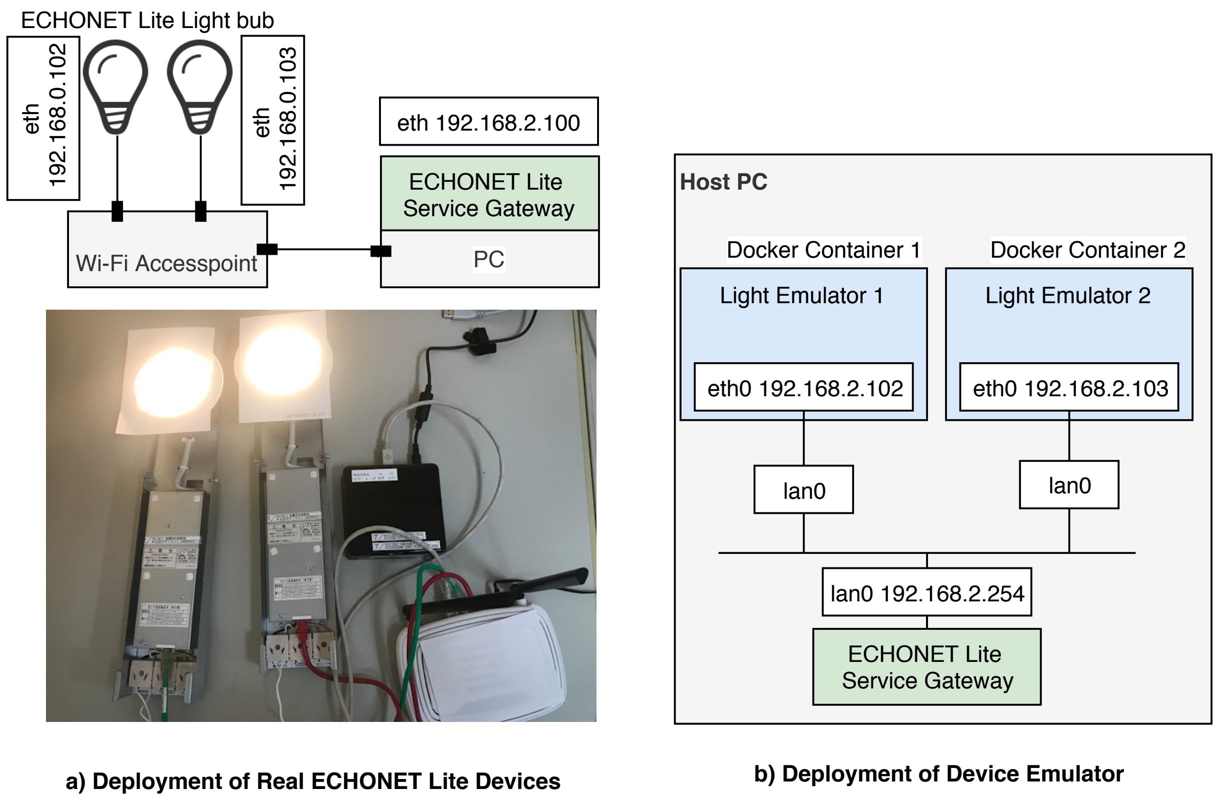


Figure 4.5: Experiment Deployment

To verify whether the simulated device can mimic commercial device operations, the implemented *ECHONET Lite HGW* in **Section** 3.2 is deployed together with commercial devices and simulated devices. The HGW is able to detect devices and enable basic interaction sequences such as *GET*, *SET*. Two commercial ECHONET Lite light bubs (Toshiba LEDD85021N-LS) as shown in Figure.4.5 **a)** is used as commercial devices. Two simulated lighting devices are configured exactly the same to commercial devices in terms of numbers of property and initial data of each property.

Table 4.3: Reponse Time Comparation Between Real and Emulated Devices

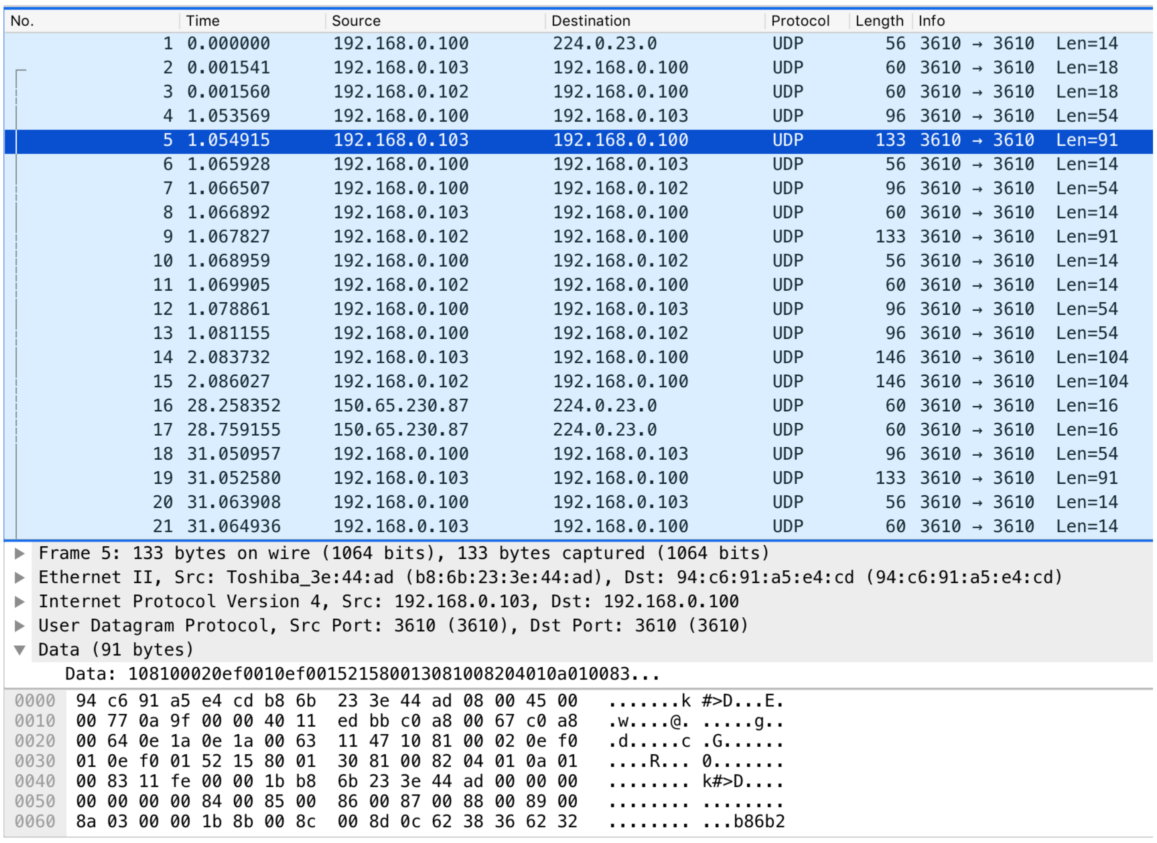
	ECHONET Lite Light 1	**ECHONET Lite Light 2**	**Emulated Light 1**	**Emulated Light 2**
Time to detect device	1067 (ms)	1080 (ms)	1027 (ms)	1048 (ms)
Time to get Profile Object	8 (ms)	7 (ms)	4 (ms)	3 (ms)
Time to get Device Object	20 (ms)	18 (ms)	12 (ms)	13 (ms)
Time to set (ON/OFF)	82 (ms)	76 (ms)	35 (ms)	36 (ms)

Table 4.3 summaries the response time for requests sent by the HGW to real and emulated devices. The necessary amount of time for the HGW to detect devices is around **1000 ms** after multi-casting the request. It requires additional **13 ms** to detect the second commercial devices and **21 ms** to detect the second emulated device. The required time to process the request of emulated devices is shorter than commercial devices because the hardware performance of emulated and commercial devices are different. The time variance is several millisecond for *GET* operation and less then 50 millisecond for *SET* operation. Obviously, the processing time of emulated devices is shorter because the processing power of the emulated device is much higher. Since the processing power of the emulated deployment environment (docker container) is adjustable, the time variance between emulated devices and commercial devices can be eliminated. However, the time variance is small enough, and in the real deployment, it is interfered with by the communication media, it could be ignored. Therefore, emulated devices are functioning in the same way as real ECHONET Lite devices.

Furthermore, packets transmitted between devices(emulated and commercial devices) and the HGW are captured as in Figure. 4.6. The result shows that emulated devices and commercial devices behaved in the same manner in responding to requests from the HGW.

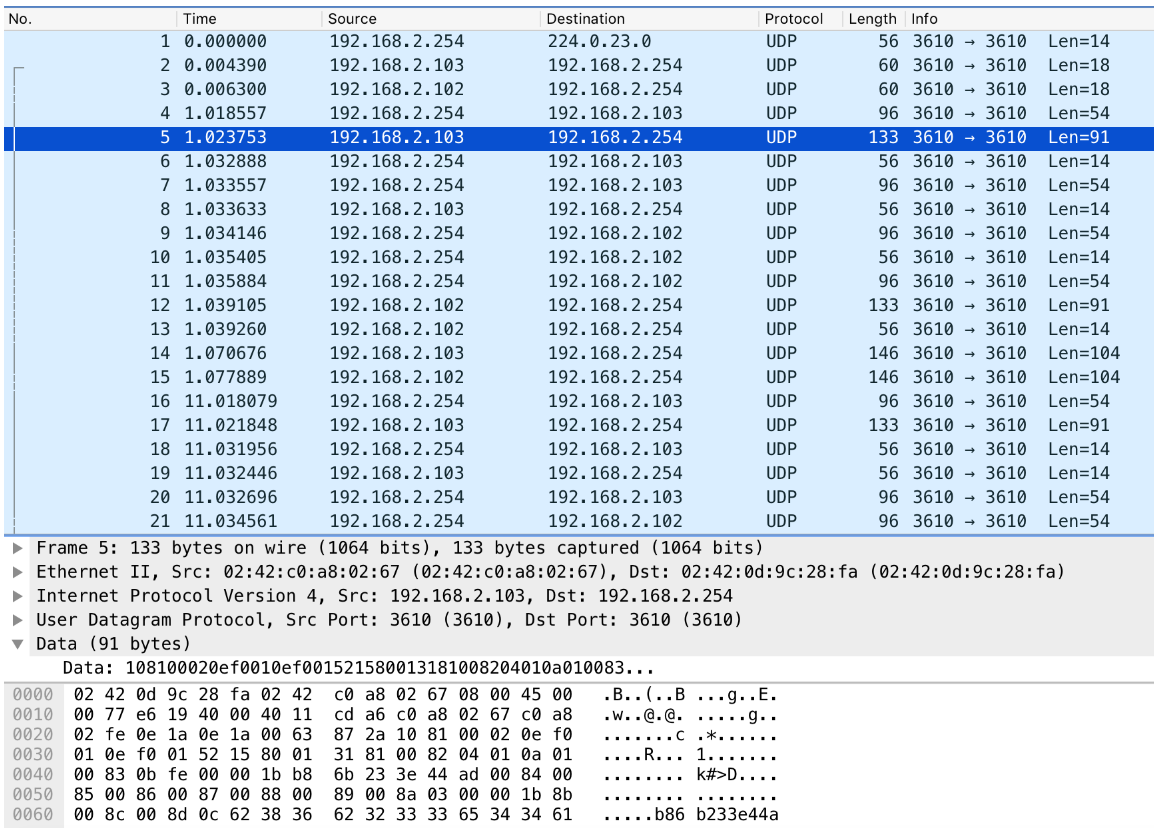
- Both of emulated devices and commercial devices replied to the node finding message request from the HGW with a **60-byte** packet.
- Both of emulated devices and commercial devices replied to the get request from the HGW with the same packet size and the same payload data.

This result proves the correctness and reliability of the DE.



```
No.    Time         Source          Destination     Protocol  Length  Info
    1  0.000000     192.168.0.100   224.0.23.0      UDP         56  3610 → 3610  Len=14
    2  0.001541     192.168.0.103   192.168.0.100   UDP         60  3610 → 3610  Len=18
    3  0.001560     192.168.0.102   192.168.0.100   UDP         60  3610 → 3610  Len=18
    4  1.053569     192.168.0.100   192.168.0.103   UDP         96  3610 → 3610  Len=54
    5  1.054915     192.168.0.103   192.168.0.100   UDP        133  3610 → 3610  Len=91
    6  1.065928     192.168.0.100   192.168.0.103   UDP         56  3610 → 3610  Len=14
    7  1.066507     192.168.0.100   192.168.0.102   UDP         96  3610 → 3610  Len=54
    8  1.066892     192.168.0.103   192.168.0.100   UDP         60  3610 → 3610  Len=14
    9  1.067827     192.168.0.102   192.168.0.100   UDP        133  3610 → 3610  Len=91
   10  1.068959     192.168.0.100   192.168.0.102   UDP         56  3610 → 3610  Len=14
   11  1.069905     192.168.0.102   192.168.0.100   UDP         60  3610 → 3610  Len=14
   12  1.078861     192.168.0.100   192.168.0.103   UDP         96  3610 → 3610  Len=54
   13  1.081155     192.168.0.100   192.168.0.102   UDP         96  3610 → 3610  Len=54
   14  2.083732     192.168.0.103   192.168.0.100   UDP        146  3610 → 3610  Len=104
   15  2.086027     192.168.0.102   192.168.0.100   UDP        146  3610 → 3610  Len=104
   16  28.258352    150.65.230.87   224.0.23.0      UDP         60  3610 → 3610  Len=16
   17  28.759155    150.65.230.87   224.0.23.0      UDP         60  3610 → 3610  Len=16
   18  31.050957    192.168.0.100   192.168.0.103   UDP         96  3610 → 3610  Len=54
   19  31.052580    192.168.0.103   192.168.0.100   UDP        133  3610 → 3610  Len=91
   20  31.063908    192.168.0.100   192.168.0.103   UDP         56  3610 → 3610  Len=14
   21  31.064936    192.168.0.103   192.168.0.100   UDP         60  3610 → 3610  Len=14
▶ Frame 5: 133 bytes on wire (1064 bits), 133 bytes captured (1064 bits)
▶ Ethernet II, Src: Toshiba_3e:44:ad (b8:6b:23:3e:44:ad), Dst: 94:c6:91:a5:e4:cd (94:c6:91:a5:e4:cd)
▶ Internet Protocol Version 4, Src: 192.168.0.103, Dst: 192.168.0.100
▶ User Datagram Protocol, Src Port: 3610 (3610), Dst Port: 3610 (3610)
▼ Data (91 bytes)
    Data: 108100020ef0010ef00152158001308100820401 0a010083...
0000  94 c6 91 a5 e4 cd b8 6b  23 3e 44 ad 08 00 45 00   .......k #>D...E.
0010  00 77 0a 9f 00 00 40 11  ed bb c0 a8 00 67 c0 a8   .w....@. .....g..
0020  00 64 0e 1a 0e 1a 00 63  11 47 10 81 00 02 0e f0   .d.....c .G......
0030  01 0e f0 01 52 15 80 01  30 81 00 82 04 01 0a 01   ....R... 0.......
0040  00 83 11 fe 00 00 1b b8  6b 23 3e 44 ad 00 00 00   ........ k#>D....
0050  00 00 00 00 84 00 85 00  86 00 87 00 88 00 89 00   ........ ........
0060  8a 03 00 00 1b 8b 00 8c  00 8d 0c 62 38 36 62 32   ........ ...b86b2
```

(a) Commercial Light Deployment



```
No.    Time         Source          Destination     Protocol  Length  Info
    1  0.000000     192.168.2.254   224.0.23.0      UDP         56  3610 → 3610  Len=14
    2  0.004390     192.168.2.103   192.168.2.254   UDP         60  3610 → 3610  Len=18
    3  0.006300     192.168.2.102   192.168.2.254   UDP         60  3610 → 3610  Len=18
    4  1.018557     192.168.2.254   192.168.2.103   UDP         96  3610 → 3610  Len=54
    5  1.023753     192.168.2.103   192.168.2.254   UDP        133  3610 → 3610  Len=91
    6  1.032888     192.168.2.254   192.168.2.103   UDP         56  3610 → 3610  Len=14
    7  1.033557     192.168.2.254   192.168.2.103   UDP         96  3610 → 3610  Len=54
    8  1.033633     192.168.2.103   192.168.2.254   UDP         56  3610 → 3610  Len=14
    9  1.034146     192.168.2.254   192.168.2.102   UDP         96  3610 → 3610  Len=54
   10  1.035405     192.168.2.254   192.168.2.102   UDP         56  3610 → 3610  Len=14
   11  1.035884     192.168.2.254   192.168.2.102   UDP         96  3610 → 3610  Len=54
   12  1.039105     192.168.2.102   192.168.2.254   UDP        133  3610 → 3610  Len=91
   13  1.039260     192.168.2.102   192.168.2.254   UDP         56  3610 → 3610  Len=14
   14  1.070676     192.168.2.103   192.168.2.254   UDP        146  3610 → 3610  Len=104
   15  1.077889     192.168.2.102   192.168.2.254   UDP        146  3610 → 3610  Len=104
   16  11.018079    192.168.2.254   192.168.2.103   UDP         96  3610 → 3610  Len=54
   17  11.021848    192.168.2.103   192.168.2.254   UDP        133  3610 → 3610  Len=91
   18  11.031956    192.168.2.254   192.168.2.103   UDP         56  3610 → 3610  Len=14
   19  11.032446    192.168.2.103   192.168.2.254   UDP         56  3610 → 3610  Len=14
   20  11.032696    192.168.2.254   192.168.2.103   UDP         96  3610 → 3610  Len=54
   21  11.034561    192.168.2.254   192.168.2.102   UDP         96  3610 → 3610  Len=54
▶ Frame 5: 133 bytes on wire (1064 bits), 133 bytes captured (1064 bits)
▶ Ethernet II, Src: 02:42:c0:a8:02:67 (02:42:c0:a8:02:67), Dst: 02:42:0d:9c:28:fa (02:42:0d:9c:28:fa)
▶ Internet Protocol Version 4, Src: 192.168.2.103, Dst: 192.168.2.254
▶ User Datagram Protocol, Src Port: 3610 (3610), Dst Port: 3610 (3610)
▼ Data (91 bytes)
    Data: 108100020ef0010ef00152158001318100820401 0a010083...
0000  02 42 0d 9c 28 fa 02 42  c0 a8 02 67 08 00 45 00   .B..(..B ...g..E.
0010  00 77 e6 19 40 00 40 11  cd a6 c0 a8 02 67 c0 a8   .w..@.@. .....g..
0020  02 fe 0e 1a 0e 1a 00 63  87 2a 10 81 00 02 0e f0   .......c .*......
0030  01 0e f0 01 52 15 80 01  31 81 00 82 04 01 0a 01   ....R... 1.......
0040  00 83 0b fe 00 00 1b b8  6b 23 3e 44 ad 00 84 00   ........ k#>D....
0050  85 00 86 00 87 00 88 00  89 00 8a 03 00 00 1b 8b   ........ ........
0060  00 8c 00 8d 0c 62 38 36  62 32 33 33 65 34 34 61   .....b86 b233e44a
```

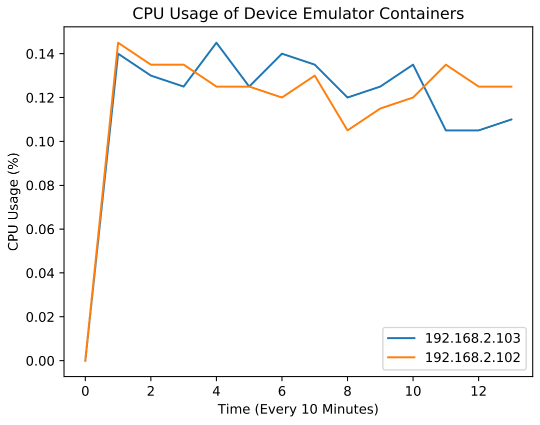
(b) Emulated Light Deployment

Figure 4.6: Devices and Home Gateway Exchanged Packets

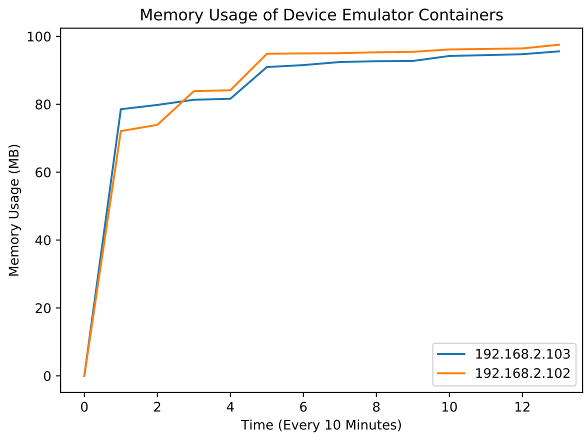
The information of the host PC in Figure. 3.20 is as bellow

- Model : Cisco UCS C200 M2 (Group L)
- CPU: Intel (R) Xeon (R) CPU X5670 (2.93 GHz/ 6 Cores)
- Number of CPU: 2
- Memory: 8GB Registered DIMM x 6 = 48GB
- HDD: 500GB x 2
- Docker Version: 17.12.0-ce, build c97c6d6
- Docker Images: Ubuntu 18.04 LTS

The CPU and memory usage of the containerized ECHONET Lite device emulator is shown in Figure. 4.8. Only one CPU core is used to deploy the experiment, and the CPU usage is about **0.14** percent each. Furthermore, an emulated device requires about **100 MB** of memory.



(a) CPU Usage



(b) Memory Usage

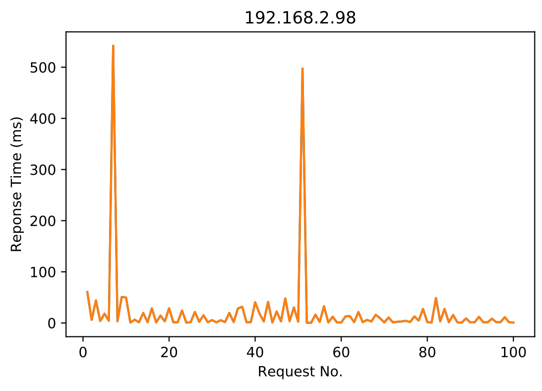
Figure 4.7: CPU and Memory Usage of Deployed Containers

Faulty Device Simulation

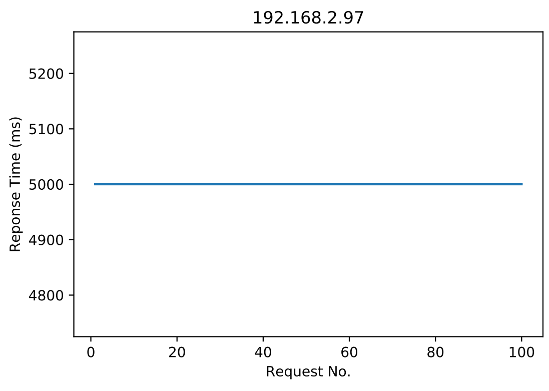
In this scenario, following emulated lighting devices have been deployed

- A normal device (**IP: 192.168.2.98**)
- A crashed device that drop 100 percent of incoming and outgoing packets (**IP: 192.168.2.97**)
- A device with response fault that does not support properties of normal device (**IP: 192.168.2.96**)
- A device with timing fault that causes a **1300 ms** delay (**IP: 192.168.2.95**)

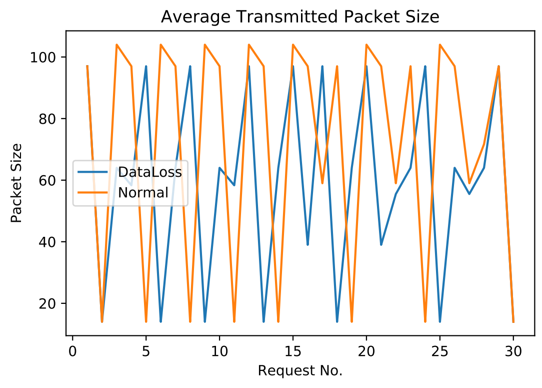
- A device with omission fault that drops 50 percent of incoming and outgoing packets (**IP: 192.168.2.94**)
- A device with drop rate **50 percent** and delay **1000 ms** (**IP: 192.168.2.111**)



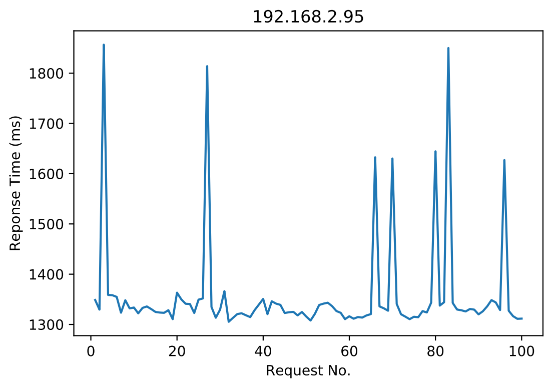
(a) Normal Device



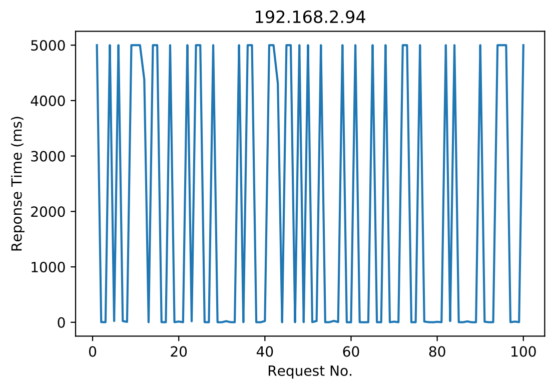
(b) Crashed Device



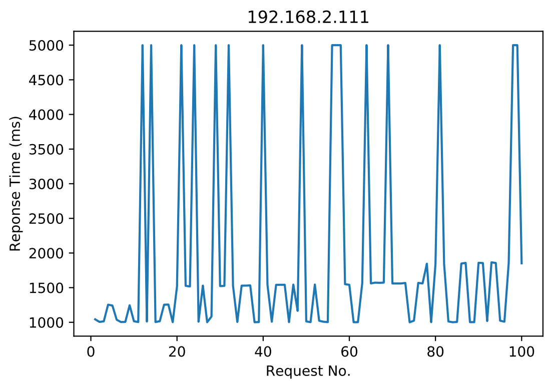
(c) Data Missing Device



(d) Timing Error Device



(e) Omission Fault Device (Drop rate 50%)



(f) Drop rate 30% and Delay 1000 ms

Figure 4.8: Faulty Device and Normal Device Comparison

The response time for the first 100 requests from the HGW of emulated devices are visualized in Figure. 4.8. The response timeout is **5000 ms** that is applied in the case of no response. In Figure. 4.8 (b), the response time is **5000 ms** for all requests which mimics the crashed fault. In Figure. 4.8 (c), the packet size of the normal device, and a device with missing data fault is visualized. Obviously, the packet size of the faulty device is smaller than the normal device. In Figure. 4.8 (d), the response time equals to **Added Delay Time (1300 ms) + Normal Response Time**. In Figure. 4.8 (e) and (f), the response timeout is reported with a rate which mimics the omission fault.

4.1.7 Conclusion

The proposed ECHONET Lite device emulator is able to simulate the behaviors of real commercial devices. Faulty devices are extended from normal devices by applying the fault model [73]. Experiment results verified the correct operations of the DE in simulating normal and faulty devices. By utilizing the docker platform, automatic and scalable deployment is achievable. The CPU usage and memory usage of the DE is about **0.15 %** and **100 MB**, respectively, for an ECHONET Lite node. Therefore, the simulation of the experiment facility with about 300 devices, namely *iHouse* in Section. 3.5.3 can be deployed in a normal PC that has 32 GB of memory. Even though the DE does not support the graphic user interface (GUI), it supports interaction with emulated devices via the file system.

4.2 Home Network Simulation

Smart home simulators and testbeds have been introduced to provide materials for researchers in the field of ambient assisted living to develop and evaluate their solutions. Since simulators reduce costs in terms of money and time, researchers are utilizing simulators to develop services to enhance user comfort, save energy, and detect abnormal behaviors in daily living activities, etc. However, available smart home simulators seem to focus on device state transitions to analyze user activities and behaviors. Simulators that can generate network traffic have not been fulfilled so far. In this section, a home network simulator that can generate network traffic of devices in the IoT area network is proposed.

4.2.1 Related Work

In [75],[76] and [77], author presented an interactive smart home simulator which provides a configurable virtual smart space for the dataset generation purposes. However, these simulators focus on providing time-series data of devices in smart homes, and from these device state transitions, a dataset for user activity recognition could be generated. It lacks to consider the network traffic from devices, as well as abnormal behaviors of devices, are not taking into consideration.

In [78], a simulated 3D smart home is proposed. It allows simulating operations and communications of devices in smart homes. It simulates devices' communication using UPnP protocol and supports a flexible mechanism to add more devices by connecting real devices into the virtual space. However, abnormal behaviors of devices are not yet considered, and it is possible to capture the traffic from real devices only.

4.2.2 Deployment Options

Essentially, there are two types of traffic which can be generated

- **Machine Generated Traffic** (MGT): the traffic generated only by the device interactions such as periodic data communication from sensors to report its measured, periodic request from HGW for network status monitoring, and so on.
- **Human Generated Traffic** (HGT): the traffic generated by the interfering of humans with devices such as turning on/ off a device, activating the human detection sensor, and so on.

Basically, network traffic is generated by having a network of devices and HGW deployed. The generated traffic is captured via network traffic capture tools such as TCPDump [4], Wireshark [5], and so on. The MGT simulation is simply done by (i) defining devices in terms of name, number, and configuration, (ii) configuring the HGW by specifying operation scenarios, (iii) deploying devices, and (iv) setting a traffic monitor to collect generated network traffic. Therefore, the MGT simulation is achievable with the DE and the proposed HGW in Section 4.1.2 and Section 3.2.

The HGT simulation is more complicated because humans involve in the loop. The simulator in [79] supports human interaction by providing an avatar so that users can control devices using the avatar. However, it is impossible to ask participants to control the avatar in order to reproduce their daily behaviors. Simulating human behaviors to support an autonomous avatar can close the gap. To build the avatar that can reflect human activity, it is required to have (i) an accurate virtual model of the target house, and (ii) simulated devices must have an accurate position as well as the area of effect. The HGT simulation requires a GUI extension for *Human Activity Simulation* of the MGT simulation.

[4] https://www.tcpdump.org/
[5] https://www.wireshark.org/

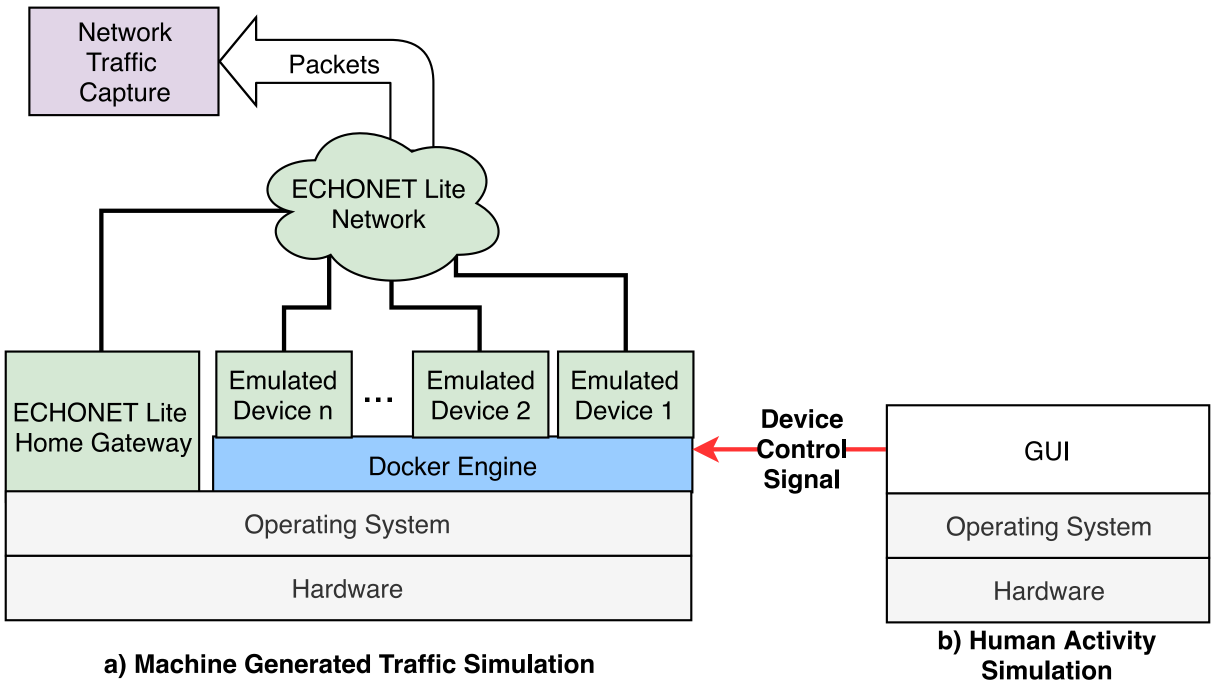
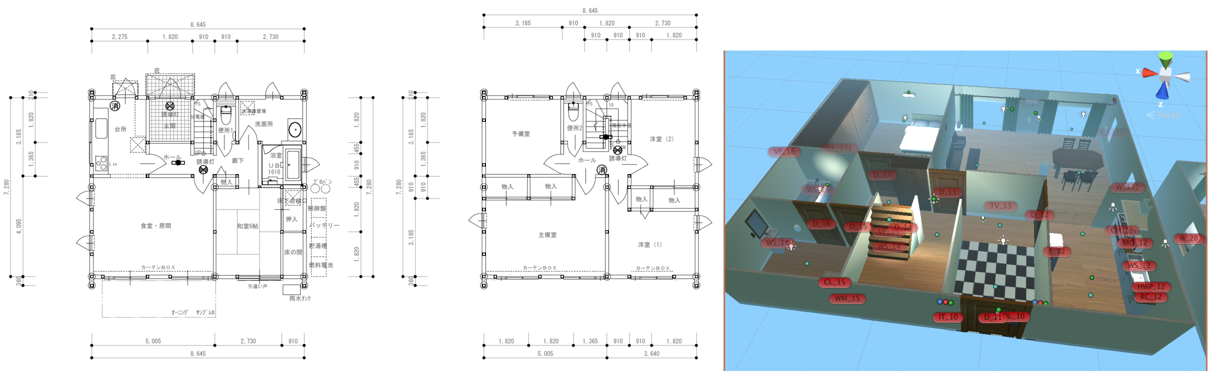


Figure 4.9: Human Generated Traffic Simulation

The overview of the MGT simulation and HGT simulation is shown in Figure. 4.9. Essentially, in the MGT simulation, the traffic is generated by the interaction between the HGW and emulated devices, and the network traffic is collected as packets by the *Network Traffic Capture* (NTC). Since a device will send a packet to notify its status changed event to the network, the HGT simulation produces network traffic by controlling device status according to the output from the *Human Activity Simulation* (HAS).



(a) Smart Home Floor Map (b) Human Activity Simulation GUI

Figure 4.10: Mapping target smart home floor map into Simulation GUI

The HAS is implementing by mapping between a target home environment floor map (physical space) into a virtual space using building information modeling (BIM) [80] technologies to ensure accurate visualization. Then, devices are deploying into the virtual environment, and these virtual devices are reflected corresponding into emulated devices of the MGT simulation (Figure. 4.9 (a)). Next, an autonomous avatar of humans is setting and moving in the virtual space. Whenever the avatar interacts with a device, the HAS sends a request to control the emulated devices based on the mapping between virtual devices and emulated devices. Since HGT is relying on human interaction, and MGT is reflecting operational aspects of networks, MGT simulation is using to generate the network traffic dataset. However, HGT is extensible from the MGT simulation and can be implemented as future work.

4.2.3 Network Traffic Dataset Generation

For the purposes of generating a network traffic dataset, the ECHONET Lite HN simulation has been deployed. Devices in the simulator are ECHONET Lite devices, and those devices are reflecting real operating devices from a testbed called the *iHouse*. Emulated device properties and initial data are loaded by values taken from the devices in the iHouse. Besides mapping real physical devices into emulated devices, faulty devices are emulated as:

- **Missing Data**: Devices that do have mandatory properties specified in the specification.
- **Delay**: Devices that have additional delay time before sending a packet. Delay time varies from **200 ms** to **4500 ms**.
- **Packet Drop**: Devices that drop the outgoing packet with a configurable rate. The drop rate varies from **5%** to **99%**.
- **Delay&PacketDrop**: Devices that suffer from both delay and packet drop faults.
- **AllErrorCombined**: Devices that suffer from all of the above faults.

The summaries of devices of the simulator is in Table 4.4.

Table 4.4: Device Configuration of the Home Network Simulator

Device Object	**Total**	**Normal Devices**	Faulty Device				
			Missing Data	**Delay**	**Packet Drop**	**Delay& Packet Drop**	**AllError Combined**
AirConditioner	**12**	6	1	1	2	1	1
AirSpeedSensor	**1**	1	0	0	0	0	0
DoorLock	**2**	1	0	1	0	0	0
ElectricCurtain	**8**	4	1	1	1	1	0
ElectricWindow	**16**	8	1	2	1	3	1
FireSensor	**2**	1	0	0	1	0	0
HotWaterPot	**2**	1	0	0	0	1	0
HumanDetectionSensor	**50**	25	2	9	4	6	4
HumiditySensor	**24**	12	1	3	3	3	2
IlluminanceSensor	**23**	11	1	3	3	3	2
InterCom	**2**	1	0	0	1	0	0
Lighting	**38**	19	2	4	4	5	4
OpenCloseSensor	**22**	11	2	1	2	4	2
Radio	**2**	1	0	0	0	1	0
Refrigerator	**2**	1	0	0	1	0	0
RiceCooker	**2**	1	0	0	0	0	1
Stove	**2**	1	1	0	0	0	0
TemperatureSensor	**24**	12	1	3	3	3	2
TV	**4**	2	0	1	1	0	0
WaterFlowRateSensor	**8**	4	0	2	2	0	0
Total	246	**123**	**13**	**31**	**29**	**31**	**19**

A total of **246** device objects are simulated by **138** ECHONET Lite nodes.
The HGW configuration is as below:

- The HGW sends a *Node Finding Message* to the multicast address of the network during start-up time.
- The HGW identifies and manages devices which replied to the *Node Finding Message*.
- The HGW sends requests to get managed device information includes a request to get the *Profile Object* and a request to get the *Device Object* at an interval of every **10 seconds**.
- The HGW sends the *Node Finding Message* to the multicast address of the network at the interval of every **2 minutes** in order to detect newly joined devices and left-the-network devices.

The HN simulator is deployed and exchanged packets of the IoT area network are captured. The network traffic dataset includes:

- The captured packets in the **pcap** format.
- IP address of devices and device information such as Device name, faulty device or not, fault category, and fault description.

- The deployment script and docker images to configure and deploy the device emulator.

- The configuration and deployment scripts of the HGW.

4.2.4 Conclusion

The HN simulator has been introduced to generate the network traffic of the IoT area network based on the ECHONET Lite protocol. The proposed simulator is able to simulate the machine-generated traffic and is extensible to support the human-generated traffic as well. The IoT network traffic dataset that includes captured packets of the deployed simulated environment is released together with device configuration and the script to regenerate the captured packets if needed. By using the HN simulator with configuration as in Table 4.4, the amount of **12674778** network packets has been collected during approximately 22 hours of deployment.

4.3 Summary

A network traffic dataset is an essential part of building AI-based solutions to provide OAM services for smart homes. Testbeds, simulations, and synthesis can generate data. However, the simulation seems to be the most appropriate solution with lower costs in terms of time and money to have the data. There have been many smart home simulations in the literature. However, they are focusing on ambient assisted living applications to improve user comfort. The IoT area network simulator is a need in order to generate network traffic data include traffic from normal devices and faulty devices. There are two primary components of the HN simulator: (i) the HGW and (ii) devices that connect to the HGW. Since the ECHONET Lite HGW architecture and implementation have been explained in chapter 3, the device simulation and the deployment of the HN simulator are explained. The device simulation supports flexible and scalable deployment. The emulated devices are operating in the same manner to commercial devices, and the faulty devices are simulated based on the fault model for the distributed system includes crash fault, omission fault, timing fault, and response fault. The proposed device emulator memory usage is less than **100 MB** and approximately **0.15%** of CPU for an ECHONET Lite node. Over 22 hours of deployment, **12674778** exchanged packets of the machine-generated traffic are captured by the HN simulator. The human-generated traffic simulation will be extended in future work.

Chapter 5

Machine Learning based Solutions for Smart Homes

By deploying the network simulator that includes the proposed HGW, normal device emulator, and faulty device emulator, network traffic data is collected in the form of raw captured packets. Data preparation is an essential part that requires to clean the input data and extract meaningful features from data.

5.1 Overview

The performance of ML algorithms heavily depends on the input data set. Raw packet capture data sets can be used to analyze the behaviors of a network for the purpose of network forensics and training. Network data that is used to build ML solutions are in the form of SNMP MIB [81], Cisco NetFlow [82], and IP Flow Information Export (IPFX) [83]. Since flow-based approaches for anomaly detection and traffic classification show the potential to achieve low time and memory overhead [84], [85], the transformation from raw capture packets into flow-based network traffic is desired.

A flow reflects the connection information between two devices, like sensors, actuators, and the HGW. Generally, a flow is identified by *source IP address, source port, destination IP address, destination port, protocol*, and all packets which share these same properties are aggregated into one flow within a time window. For example, a complete flow in an online youtube video streaming section consists of all packets sent from the host to the youtube server. It includes session setup, data exchange, and session finish. *Netflow* [82] aggregates these flows into one single flow record until an active or inactive timeout is reached.

Table 5.1: Typical Attributes of a flow record with an example

Attribute	**Data Type**	**Example**
Start Time	timestamp	2019-10-30 21:12:23.289
Duration	time	0.389 second
Protocol	categorical	UDP
Source IP Address	categorical	192.168.2.254
Source Port	numeric	3610
Destination IP Address	categorical	192.168.2.100
Destination Port	numeric	3610
Bytes	numeric	128
Packets	numeric	2

Table 5.1 shows the typical attributes of an **unidirectional** flow record. The exemplary values for an ECHONET Lite flow record where a device with the IP address *192.168.2.254* sent *two packages* of a total of 128 bytes to a device with the IP address *192.168.2.100)* during the period of *389 ms* are also included.

Unlike the unidirectional flow represents packets flowing in one direction only, a bidirectional (Biflow) [86] represents packets flowing on two directions between endpoints on a network. The *Biflow* more accurately describes behaviors and gives more insight information of a network system [86], [87].

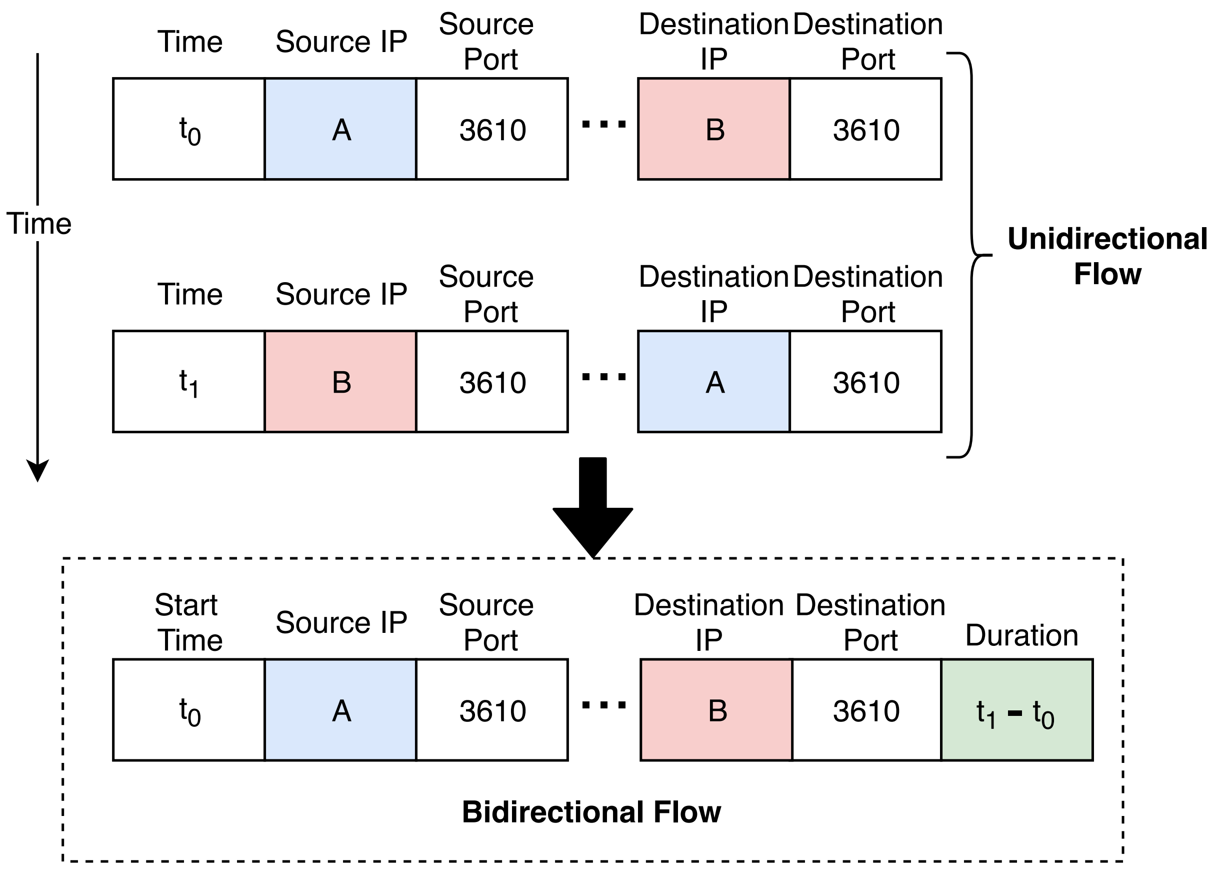


Figure 5.1: Bidirectional Flow Conceptual Diagram

The conceptual diagram of mapping from unidirectional flows, which have the same value of 5-tuple of *source IP address, source port, destination IP address, destination port, protocol* but with the reversion of the source address and the destination address, is shown in Figure. 5.1. The direction of a *Biflow* can be decided by the time order or by the client-server relation based on endpoint identifications. The bidirectional flow record provides information about client-server interactions.

5.2 Related Work

A network traffic flow generator, namely *NetFlowMeter* [88], [89], has been widely applied to generate bidirectional flows of network traffic data sets such as data set for android malware [90], data set for DDos attack detection [91], data set for intrusion detection [92], and so on.

The *NetFlowMeter* is a Java-based open-source bidirectional flow generator that takes the capture packets from a pcap file or performs the live packet capture and aggregates those packets into a list of *Biflow* in the timely order. The *NetFlowMeter* is able to export **83** features of a flow such as *Number of packets in forward direction, Number of packets in backward direction, interaction duration*, etc. The *NetFlowMeter* support both UDP and TCP flows while UDP flows are terminated by the flow timeout, TCP flows are terminated by the FIN packet. Apparently, the flow timeout can be applied for both TCP and UDP flow by an adjustable value. However, the situation is different in the IoT Area Network where devices are usually low-power devices with the wake-up mechanism. It means that for power saving purposes, devices sleep all the time and wake-up to answer requests, then quickly terminate the session ((in seconds, milliseconds)) and switch to the sleep mode. While video streaming or gaming devices where the session is long and devices keep the connection for a long period of time (in minutes, hours). Moreover, the connection direction of the *NetFlowMeter* is decided by a timely order only. However, in the IoT Area Network, it is required to have the direction decided by the initiator (the HGW). Therefore, a network flow calculator for the IoT Area Network is desired.

5.3 Network Flow Calculator

The overview of the proposed network flow calculator (*flowCal*) for the IoT Area Network is illustrated as in Figure. 5.2.

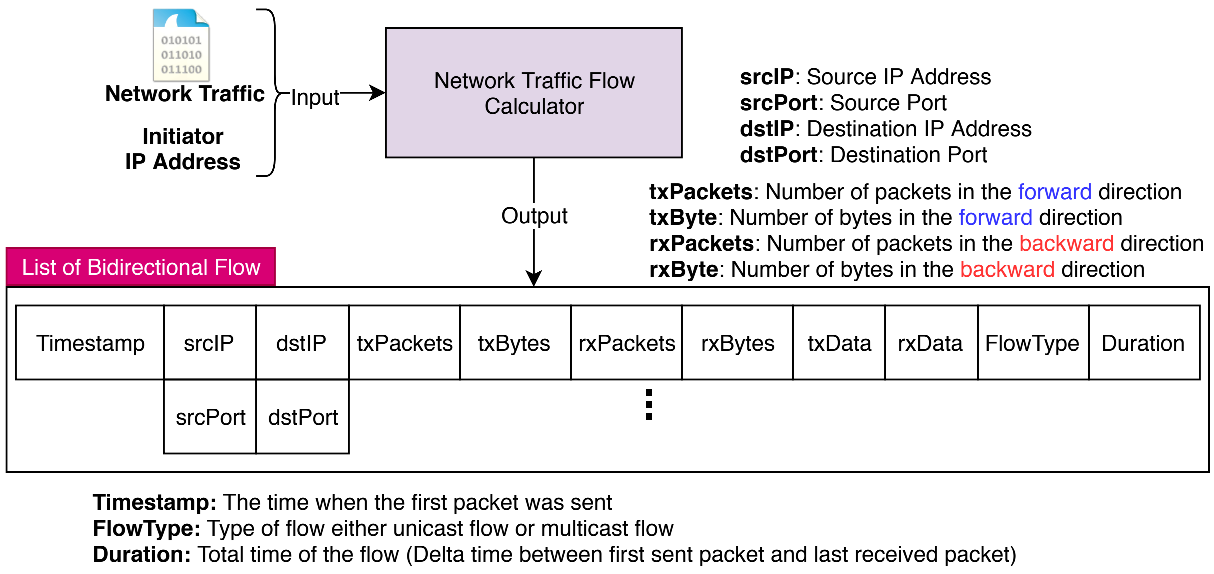

010101
011010
011100
Network Traffic
Initiator
IP Address
Input
Network Traffic Flow Calculator
srcIP: Source IP Address
srcPort: Source Port
dstIP: Destination IP Address
dstPort: Destination Port
Output
txPackets: Number of packets in the forward direction
txByte: Number of bytes in the forward direction
rxPackets: Number of packets in the backward direction
rxByte: Number of bytes in the backward direction
List of Bidirectional Flow
Timestamp
srcIP
dstIP
txPackets
txBytes
rxPackets
rxBytes
txData
rxData
FlowType
Duration
srcPort
dstPort
Timestamp: The time when the first packet was sent
FlowType: Type of flow either unicast flow or multicast flow
Duration: Total time of the flow (Delta time between first sent packet and last received packet)

Figure 5.2: Network Traffic Flow Calculator Overview

The *flowCal* takes the network traffic in the form of captured packets as the input and extracts unidirectional flows. Then, bidirectional flows are aggregated from extracted unidirectional flows and the *IP address of initiator* to determine the direction of bidirectional flows. Since a session is usually initiated by the HGW and the direction of bidirectional flows are assigned by the initiator, the *IP address of initiator* is the IP address of the HGW. Furthermore, besides sending unicast requests to target devices in order to collect device sources, the HGW sends multicast requests in order to detect newly joined devices or expired devices (as stated in Section 3.3.3). Therefore, there are two types of flow: the *multicast* flow and the *unicast* flow. Basic attributes of a bidirectional flow that is exported using the *flowCal* is as in Table 5.2.

Table 5.2: Basic attributes of exported bidirectional flows with an example

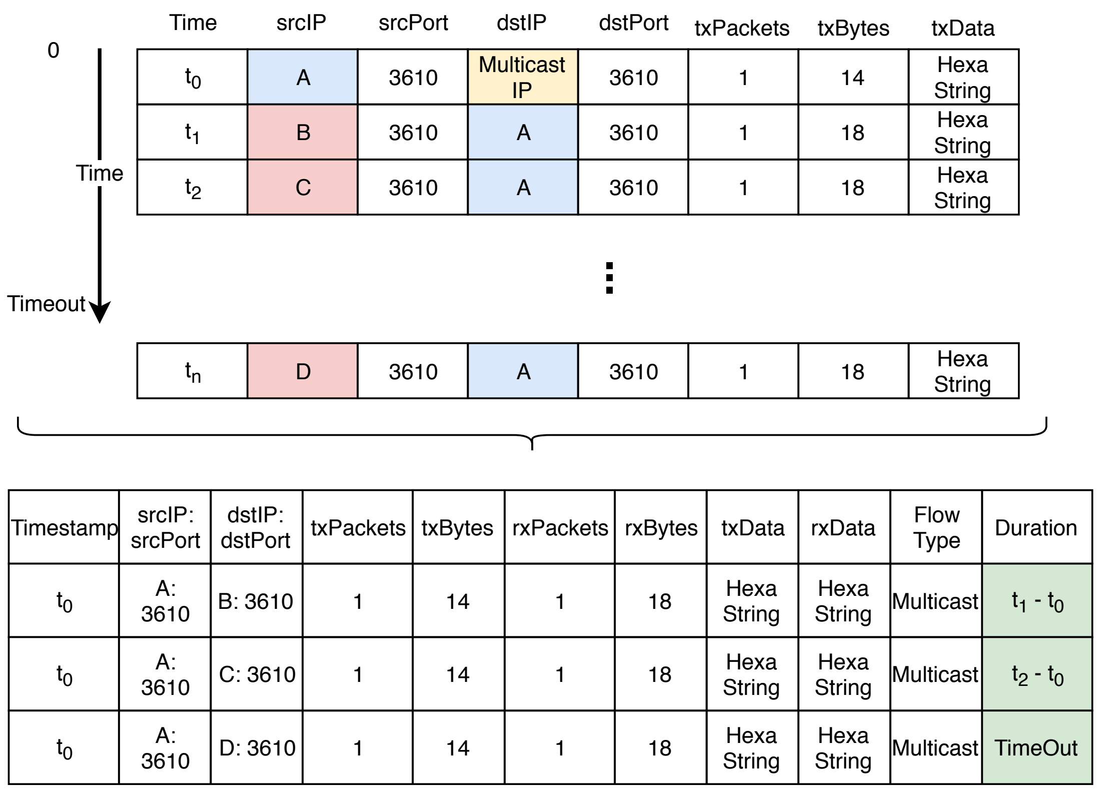
Attribute	**Short name**	**Data Type**	**Example**
Timestamp	Timestamp	Time	2019-10-30 21:12:23.289
Source IP Address	srcIP	String	192.168.2.254
Source Port	srcPort	Integer	3610
Destination IP Address	dstIP	String	192.168.2.103
Destination Port	dstPort	Integer	3610
Sent Packet Count	txPackets	Integer	2
Sent Bytes	txBytes	Integer	68
Received Packet Count	rxPackets	Integer	2
Received Bytes	rxBytes	Integer	88
Sent Data	txData	Hexa String	String in hexa format
Received Data	rxData	Hexa String	String in hexa format
Flow Type	FlowType	Boolean	Unicast
Duration	Duration	Float	348.3 ms

The **Timestamp** is the time when the first packet has been sent, and the *Duration* is the delta time to complete the session (delta time between the time of the first sent packet and the time of the last received packet). Other attributes which are derivable from basic attributes such as

- Minimum size of sent packets
- Maximum size of sent packets
- Mean size of sent packets
- Standard deviation of size of sent packets
- Minimum size of received packets
- Maximum size of received packets
- Mean size of received packets
- Standard deviation of size of sent packets

could be calculated and included in the output.

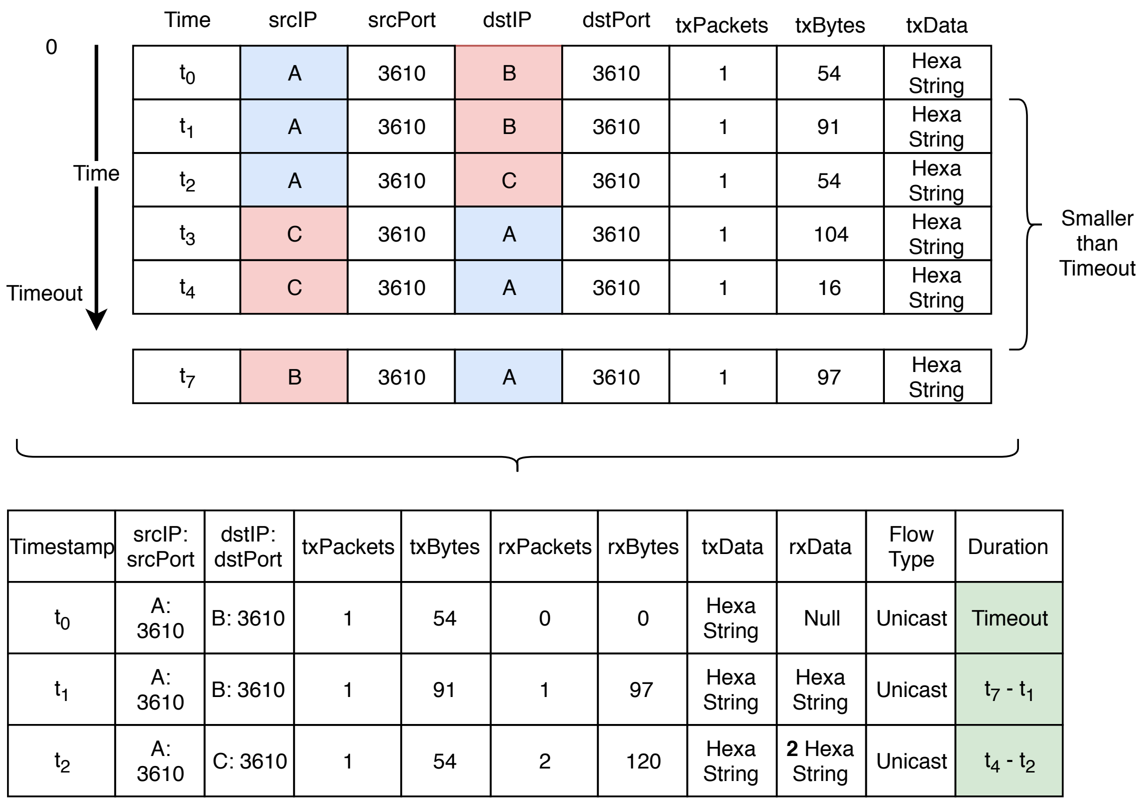
The solution to aggregate **multicast** and **unicast** flows is visualized in Figure. 5.3 and Figure. 5.4 respectively.



Time	srcIP	srcPort	dstIP	dstPort	txPackets	txBytes	txData
t_0	A	3610	Multicast IP	3610	1	14	Hexa String
t_1	B	3610	A	3610	1	18	Hexa String
t_2	C	3610	A	3610	1	18	Hexa String
⋮							
t_n	D	3610	A	3610	1	18	Hexa String

Timestamp	srcIP: srcPort	dstIP: dstPort	txPackets	txBytes	rxPackets	rxBytes	txData	rxData	Flow Type	Duration
t_0	A: 3610	B: 3610	1	14	1	18	Hexa String	Hexa String	Multicast	t_1 - t_0
t_0	A: 3610	C: 3610	1	14	1	18	Hexa String	Hexa String	Multicast	t_2 - t_0
t_0	A: 3610	D: 3610	1	14	1	18	Hexa String	Hexa String	Multicast	TimeOut

Figure 5.3: Bidirectional Multicast Flow Aggregation



Time	srcIP	srcPort	dstIP	dstPort	txPackets	txBytes	txData
t_0	A	3610	B	3610	1	54	Hexa String
t_1	A	3610	B	3610	1	91	Hexa String
t_2	A	3610	C	3610	1	54	Hexa String
t_3	C	3610	A	3610	1	104	Hexa String
t_4	C	3610	A	3610	1	16	Hexa String
t_7	B	3610	A	3610	1	97	Hexa String

Timestamp	srcIP: srcPort	dstIP: dstPort	txPackets	txBytes	rxPackets	rxBytes	txData	rxData	Flow Type	Duration
t_0	A: 3610	B: 3610	1	54	0	0	Hexa String	Null	Unicast	Timeout
t_1	A: 3610	B: 3610	1	91	1	97	Hexa String	Hexa String	Unicast	t_7 - t_1
t_2	A: 3610	C: 3610	1	54	2	120	Hexa String	**2** Hexa String	Unicast	t_4 - t_2

Figure 5.4: Bidirectional Unicast Flow Aggregation

5.4 Experiment

5.4.1 Data Processing and Labeling

The aggregated bidirectional flows are able to reflect device behaviors. As statistical, the more flows collected, the better judgment of device behaviors. Therefore, we can cluster flows of a device for a day or a week for the training data set. However, in real-world deployment, to make a prediction using the model trained with the previous data, it is required to collect the same numbers of flow in real-world deployment, and it creates a huge delay time in collecting data for the adjustment. This section discusses how to choose numbers of observations and numbers of features from an observation that fully reflects device behaviors in an appropriate time window.

Apparently, devices in the IoT Area Network perform periodical tasks such as periodically sending device information, periodically querying device status, the machine-generated traffic contains the repeated patterns. Since we can assume that the flows are independent for each repeated period, a cluster of flows, which are collected during a repeated period, is used as a data unit to train the model and also to predict device behavior even though in the real deployment. For example, in the data set collected by deploying the HN simulator in Section 4.2.3, the HGW periodically multicasts the *Node Finding Message* at the rate of **2** minutes. Therefore, all extracted flows between two multicast requests could be clustered into a vector that reflects device behavior. Furthermore, in a real deployment, it takes a time of **2** minutes to collect flows from the target device in order to make the judgment.

A flow contains 11 attributes as in Table 5.1, and features that characterize devices are extracted from these attributes. Since *Timestamp, Source IP address, Source port, Destination IP address, and Destination port* are trivial attributes, remaining 8 attributes *Sent Packet Count, Sent Bytes, Received Packet Count, Received Bytes, Sent Data, Received Data, Flow Type, and Duration* are extracted as features and organized as an input vector to represent for devices in investigating ML methods.

The dimension of the input vector is calculated by the Equation 5.1.

$$\textbf{Input Vector Dimension} = \textbf{Numbers of Flow} \times \textbf{Numbers of Selected Features} \tag{5.1}$$

Because the input vector represents a device identified by the IP address, it can be labeled using data set descriptions described device configurations.

The conceptual diagram which summaries **Input** and **Output** of the pre-processing process is illustrated in Figure 5.5.

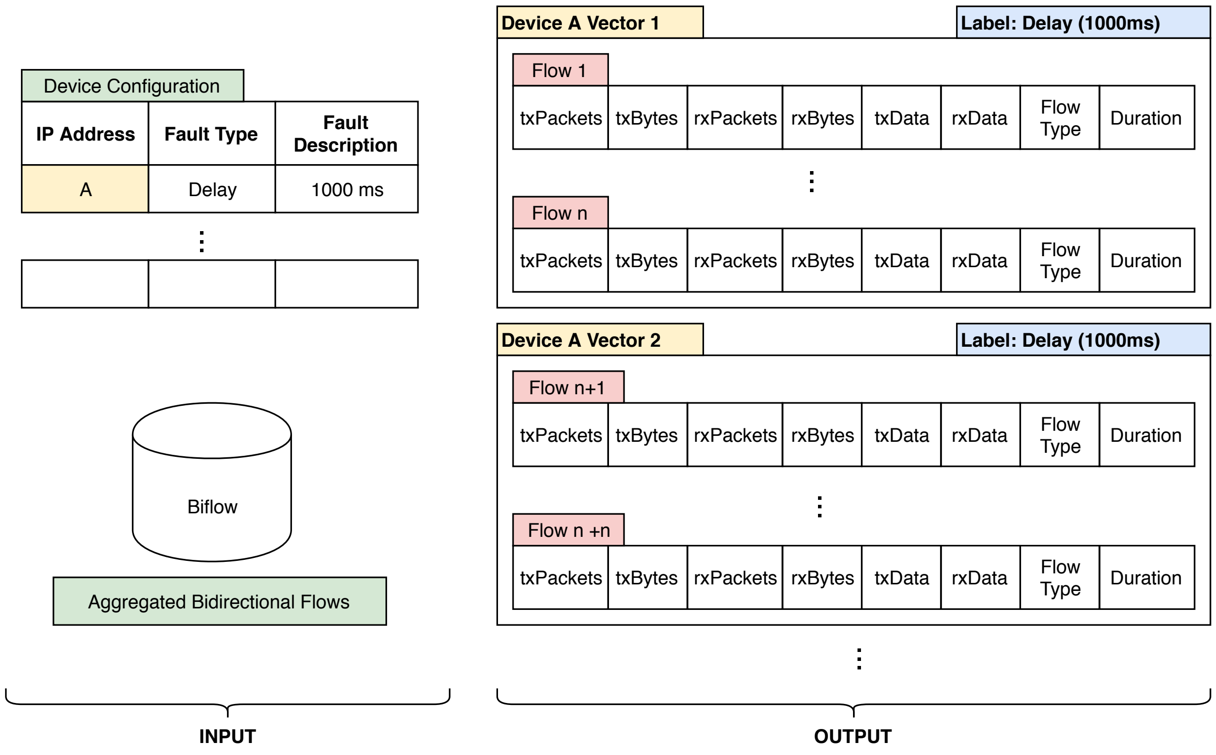


Figure 5.5: Data Pre-Processing Conceptual Diagram

5.4.2 Results

The packet capture data generated in Section is used as the data set in this experiment. The *.pcap* file of **12674778** packets which collected during over the period of 22 hours is extracted into bidirectional flows by utilizing the network flow calculator in Section 5.3. The HGW is configured to send the multicast request at an interval of **2** minutes, and send to unicast to each device at an interval of **10** seconds. The maximum number of multicast flows and unicast flows which are used as input data is as in equation 5.2 and equation 5.3 respectively.

$$138(\textbf{node}) \times 30(\textbf{request/hour}) \times 22(\textbf{hours}) = 91080(\textbf{multicast flows}) \tag{5.2}$$

$$138(\textbf{node}) \times 360(\textbf{request/hour}) \times 22(\textbf{hours}) = 1092960(\textbf{unicast flows}) \tag{5.3}$$

During the period of 2 minutes in between two multicast requests, the maximum number of flows of a node is following the equation 5.4

$$(1(\textbf{node}) \times 6(\textbf{request/minute}) \times 2(\textbf{minutes})) + 1 = 13(\textbf{flows}) \tag{5.4}$$

However, it might not be possible to collect **13** flows from faulty devices since they may drop some requests. Some ML techniques such as SVM and ANN require the input as a fixed-length vector. Therefore, we need to pad dumb flows, which contain all zero values to the devices so that all the devices have the same number of flows. Since all bidirectional flows collected from the period of **2** minutes are combined to make a sample to train the model, each device will have **660** samples, and the total numbers of the data set are **91080** sample. We split 80% of samples (72864 samples) into the training set and the rest of 20% into the testing set (18216 samples). Each sample is represented by a vector with $8(features/flow) \times 13(flow) = 104$ dimensions.

Data Normalization

Data normalization is an essential step during data preparation to scale the data and speed up the training process [93]. In general, there are two common approaches in data normalization includes:

- Min-Max Normalization: To scale all features of a sample to the same scale. However, it is not good tot handle outliers.
- Z-Score Normalization: It is good to handle outliers. However, it does not guarantee all features have the same scale.

As features from the bidirectional flows are not in the same in nature (e.g. packet count and packet size), Z-Score Normalization is used to normalize samples of the data set. The formula for Z-Score normalization is in equation 5.5

$$\frac{Value - Mean}{StandardDeviation} \tag{5.5}$$

By the Z-Score normalization, if a value is greater than *mean*, it will be a positive number. If it is smaller than *mean*, it will be a negative number. If it is equal to *mean*, it will be normalized to 0.

Data Distribution Result

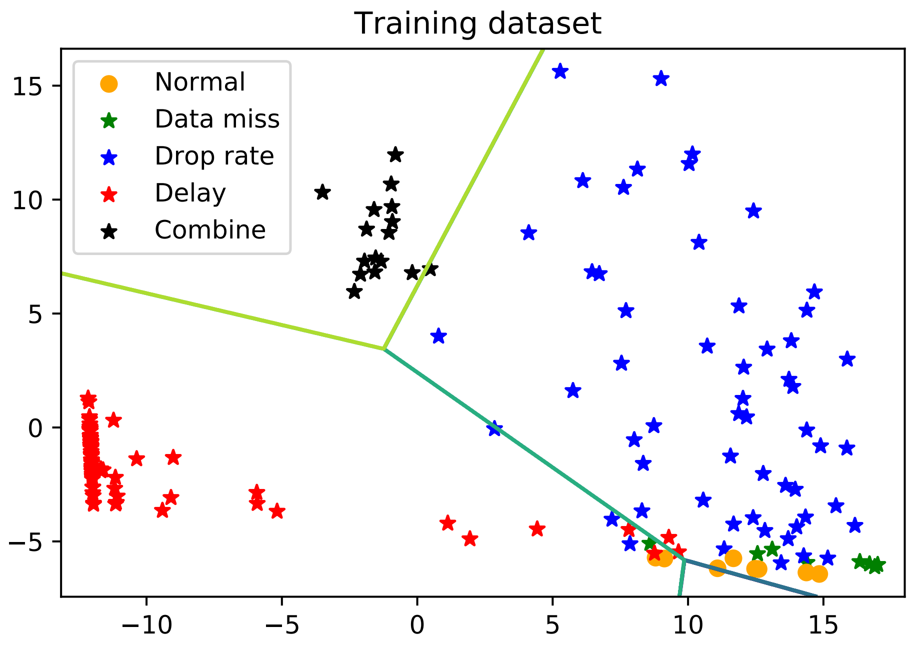


Figure 5.6: Data Distribution with PCA

Because a sample is represented by a **104**-dimension vector, it is impossible to visualize the data distribution. However, the Principal Component Analysis (PCA) [94] could be used to reduce the dimension of data. PCA is a simple and popular method to reduce dimension and avoid information loss from data [95]. The PCA is apply to reduce the **104-dimension** vector into **2-dimension** vector in order to visualize data distribution. Some samples are selected and visualized as in Figure. 5.6. Data points of devices that have a delay (timing fault) are separated into the points of normal devices. We can use a line to separate the delay samples from the normal samples. The distances of the delay samples respect to its delayed time. The data points of devices have the omission fault (packet drop) are mixed up with normal devices.

5.5 Summary

This section describes the way to build ML-based solutions for the IoT Area Network in smart homes. The data preparation is an essential part that requires to clean the input data and extract features related to problems from a massive amount of data. The network flow calculator, namely *Flowcal*, has been proposed in order to extract bidirectional flows from capture packets that can collect by deploying the HN simulator. The*Flowcal*, which is customized for the IoT Area Network, is able to appoint the flow initiator and also handle multicast flows.

The *Flowcal* is applied to extract bidirectional flows from the data set provided in Section 4.2.3. Then, data sample to train and test the ML-based models has been calculated. A data sample is a 104-dimension vector that includes **eight** features from each flow combined with 13 flows for each multicast cycle. The total numbers of **91080** samples that represent for 138 nodes (384 objects) during the 22 hours of deployment are collected. Those samples are split into training data set and testing data set by the 80%(training):20%(testing) ratio. An example of data distributions is visualized by applying the PCA to reduce the data dimension. It showed that data points of devices have the omission fault are mixed up with normal devices.

Chapter 6

Application: Machine Learning Based Network Traffic Classification

Since detecting problems is a starting point to build an automated OAM solution and network traffic data could be used to diagnose the health of a smart home network, a network traffic classification application for anomaly detection is implemented based on data generated from the proposed framework to verify the usability of the generated data

6.1 Machine Learning Methods

In this dissertation, we investigate three machine learning methods: **Decision Tree**, **Support Vector Machine**, and **Artificial Neural Network**.

- Decision Tree is widely used to demonstrate the rule-based approach [96].
- Support Vector Machine is the best linear classifier approach [97].
- Artificial Neural Network is a general non-linear classifier that achieves huge success in many real-world problems [98].

6.1.1 Decision Tree

The Decision Tree (DT) is a tree-like model that supports decision-making processes. DT belongs to the category of supervised learning mostly used for classification. The goal of DT is to generate a model that can classify the value of a target sample by learning decision rules from extracted features of the training data. In the DT model, data samples flow from the root through internal nodes to the leaves. Leaves are the final prediction of the class. Each node is a condition to check the target sample. The condition is tested on a feature of the target sample at a node. The result of the checking process drives the sample to the next node until reaching a leaf. Because conditions on attribute are interpretable, they can be used as rules to classify the data samples. An example of using DT for decision making is illustrated in Figure. 6.1

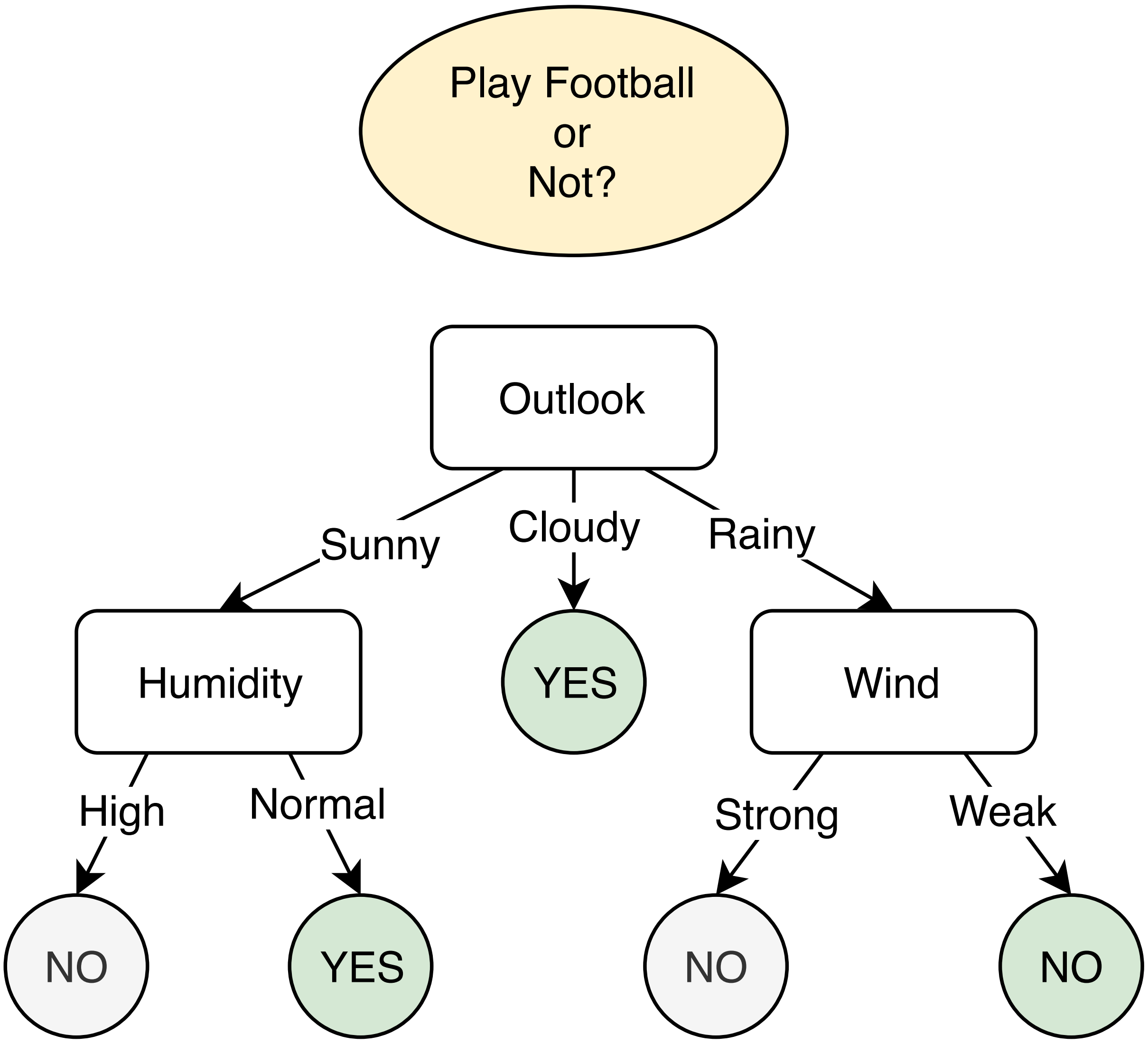


Figure 6.1: An example of applying Decision Tree for decision making

The advantages of DT [96] are:

- Easy to understand and envisage the tree structure.
- Data preparation processes such as normalization, dummy padding are not required.
- Support multiple outputs classification.
- Results are easily explainable and interpretable since DT uses the white-box model.

Some disadvantages of DT are:

- Easy to overfit with a data set.
- DT is not stable as a small variation in the data set might change the tree completely.
- DT requires a balanced data set in order to deal with the bias.
- DT does not generalize well to unseen samples when deploying to the real environments.

6.1.2 Support Vector Machine

The Support Vector Machine (SVM) [99] is originally a binary classifier that tries to find a decision boundary that separates two categories in the data set. The decision boundary determined by SVM has the largest margin to the closest data points of both categories. For the multi-class problem, techniques such as **one-against-all** [100], **pairwise**, **all-at-once** [101] and **error-correcting-output**[102] could be used. In this dissertation, one-against-all is used for linear SVM classifier. In the case of linear SVM, the decision boundaries are hyperplanes. The predictions produced by linear SVM have higher confidences than other hyperplanes found by other linear classification methods. For the case of binary linear SVM, the predicted label is the sign of the linear combination of: **Data Sample X** $=< x_1, x_2, x_3... >$ and **Weight W** $=< w_1, w_2, w_3... >$. The weight represents for the hyperplane as the decision boundary and the margin is $\frac{1}{|w|}$. We can assume that the label **y** $\in$ **-1,1** in the case of binary linear SVM. The prediction is correct if it has the same sign with the ground truth label in equation 6.1:

$$y_k(w \times x_k) >= 1 \tag{6.1}$$

SVM tries to find **w** that separates the data points completely and has the largest margin as in equation 6.2.

$$\min(\frac{|w^2|}{2}) \;\; s.t. \;\; \forall k, y_k(w \times x_k) >= 1 \tag{6.2}$$

In the case that the data set is inseparable, the soft margin technique could be applied in order to tolerate the separation errors. For the multi-class problem with **n** class, we will need **n** hyperplanes in the one-against-all manner

The advantages of the SVM [103] include:

- Work well in high dimensional spaces
- Achieve the memory efficient as it uses support vectors as inputs for the decision function.
- Achieve flexibility in choosing or customizing kernel functions.

Some disadvantages of SVM are:

- It causes over-fitting easily if the number of features is much greater than the number of samples.
- It requires proper configurations of several key parameters to achieve a satisfactory result.

6.1.3 Artificial Neural Network

The Artificial Neural Network (ANN)[104] is a biologically-inspired programming paradigm which learns from observational data. ANN uses the examples to infer rules for determining the classes automatically. The conceptual diagram of the AAN is visualized in Figure. 6.2.

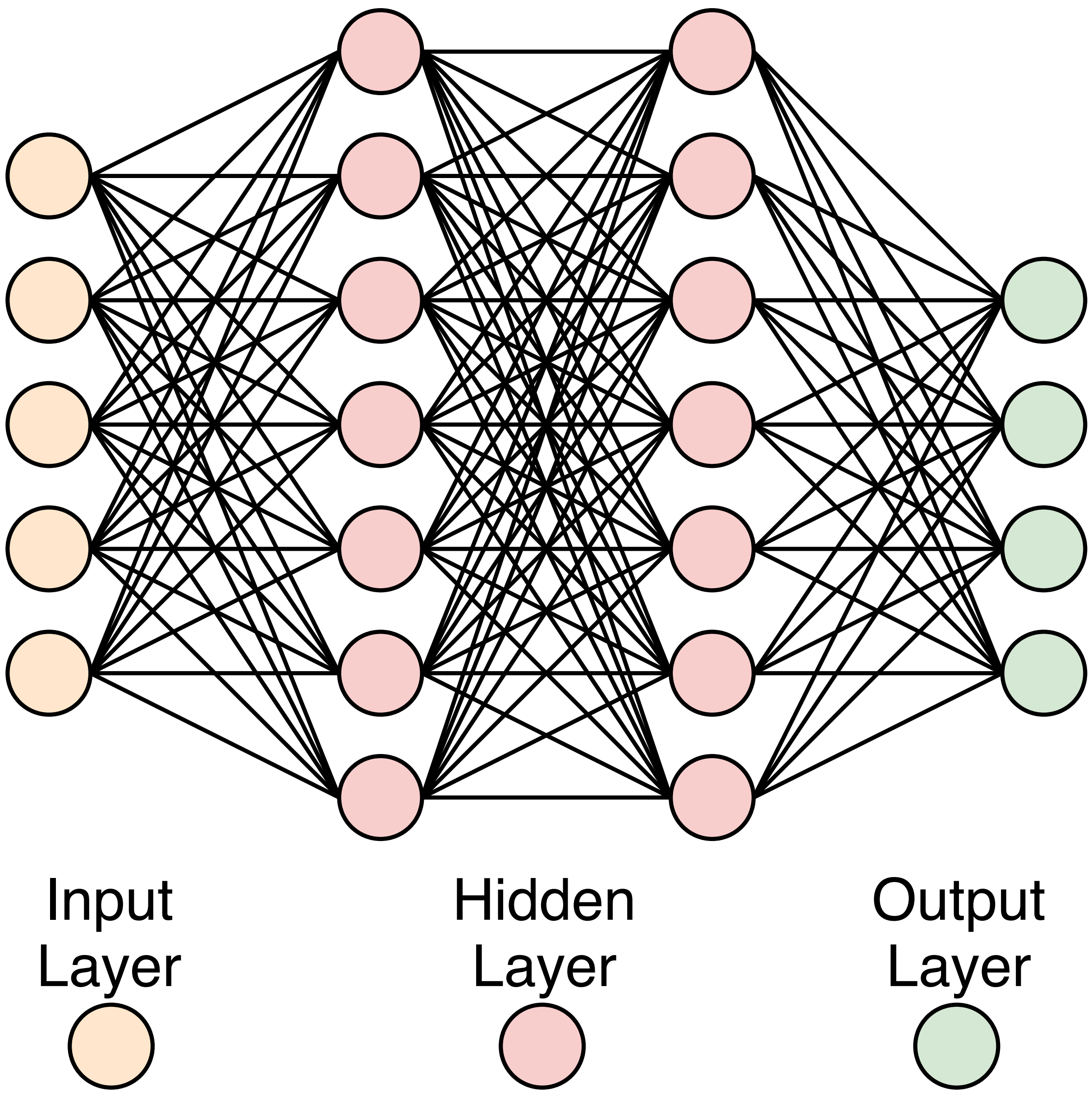


Figure 6.2: Conceptual Diagram of the Artificial Neural Network

The ANN contains multiple layers of interconnecting nodes. An ANN has three main layers: one input layer, one output layer, and one or more hidden layers in the middle. The first layer is the input layer, which takes the value of the input data sample. The last layer is the output layer, which is the predictions of the ANN. The hidden layer $\mathbf{h}_i$ take the output of its previous layer $\mathbf{h}_{i-1}$, multiplies with the weight $\mathbf{w}_i$ and then produces output by an activation function σ as in equation 6.3

$$h_i = \sigma(z_i) = \sigma(w_i \times h_{i-1}) \tag{6.3}$$

We can view ANN as a function **f'** that tries to approximate the real hidden underlying a function **f** mapping the data point to its class. The function **f'** is parameterized by the weights of the hidden layers of ANN. A loss function is applied to measure the distance between the predicted results and the ground truths. The training process of the ANN means minimizing this loss function. Theoretically, an ANN can approximate any continuous function of the input space. Hence, the ANN is chosen as a generalized method to demonstrate the non-linear classifier problems.

Some advantages of ANN include:

- Is able to deal with non-linear classifier problems
- Is able to work with incomplete information

The disadvantages of AAN are:

- It is unable to explain results interfered by the ANN.
- Different random weight initialization may lead to different performances.

6.2 Experiment

In this section, we investigate the capabilities of machine learning on predicting whether a device from the IoT Area Network is a faulty device or not based on the traffic captured from the home network simulator. The ML techniques such as Decision Tree, Support Vector Machine, and Artificial Neural Network are investigated on predicting the behaviors of the devices. This study aims to stimulate and predict the unseen possible abnormal behaviors of the devices deploying in smart homes.

6.2.1 Results

All experiments are conducted based on the previous data using the computer infrastructure of startBED Hokuriku [1]. The configuration information of the computer that uses to train and test ML models during the experiment is as

- Model : Cisco UCS C200 M2 (Group L)
- CPU: Intel (R) Xeon (R) CPU X5670 (2.93 GHz/ 6 Cores)
- Number of CPU: 2
- Memory: 8GB Registered DIMM x 6 = 48GB
- HDD: 500GB x 2
- Operating System: Ubuntu 16.04.6 LTS
- Python version 3.6

The average accuracy of predicting six classes of traffic includes **Normal, Response Fault, Timing Fault, Omission Fault, Combined of Timing and Omission Fault, and All Fault Combined** after running ten times **without** data normalization is:

- Decision Tree : 93.23%

[1] http://starbed.nict.go.jp/en/equipment/index.html

- Support Vector Machine : 66.30%
- Artificial Neural Network: 91.86%

and **with** data normalization is:

- Decision Tree : 93.33%
- Support Vector Machine : 89.64%
- Artificial Neural Network: 96.72%

Decision Tree

Table 6.1: Decision Tree Without Normalization Confusion Matrix

Ground Truth	Normal	Response Fault	Timing Fault	Omission Fault	Combined of Timing and Omission	All Fault Combined
Normal	9030	11	1	135	0	0
Response Fault	6	1052	0	6	0	0
Timing Fault	1	0	2013	86	158	3
Omission Fault	242	8	7	1819	32	20
Combined of Timing and Omission	1	0	55	83	2085	170
All Fault Combined	0	0	2	71	143	1114

Table 6.2: Decision Tree With Normalization Confusion Matrix

Ground Truth	Normal	Response Fault	Timing Fault	Omission Fault	Combined of Timing and Omission	All Fault Combined
Normal	9025	11	2	139	0	0
Response Fault	5	1054	0	6	0	0
Timing Fault	16	0	2028	71	146	0
Omission Fault	243	8	6	1815	34	22
Combined of Timing and Omission	1	0	56	76	2093	169
All Fault Combined	0	0	3	67	144	1117

Results of the decision tree on the testing data set without and with data normalization are summarised in Table 6.1, and Table. 6.2. Basically, data normalization did not improve the accuracy much. Without data normalization, the accuracy of predicting normal device is **98.40%**, device with response fault is **98.97%**, device with timing fault is **89.70%**, device with omission fault is **85.30%**, device with timing and omission fault is **87.40%**, and device with all faults is **83.92%**. After the data normalization, the accuracy of predicting normal device is **98.34%**, device with response fault is **99.06%**, device with timing fault is **89.69%**, device with omission fault is **85.29%**, device with timing and omission fault is **88.09%**, and device with all faults is **83.98%**.

Support Vector Machine

Table 6.3: Support Vector Machine Without Normalization Confusion Matrix

Ground Truth	Normal	Response Fault	Timing Fault	Omission Fault	Combined of Timing and Omission	All Fault Combined
Normal	8248	314	2	613	0	0
Response Fault	152	904	0	9	0	0
Timing Fault	378	0	1252	586	94	26
Omission Fault	1202	95	125	586	94	26
Combined of Timing and Omission	717	95	345	209	902	127
All Fault Combined	354	0	220	119	360	277

Table 6.4: Support Vector Machine With Normalization Confusion Matrix

Ground Truth	Normal	Response Fault	Timing Fault	Omission Fault	Combined of Timing and Omission	All Fault Combined
Normal	8905	50	28	188	5	1
Response Fault	32	1031	1	0	0	0
Timing Fault	0	0	1981	12	142	23
Omission Fault	244	74	137	1524	76	74
Combined of Timing and Omission	29	47	155	105	1932	127
All Fault Combined	4	0	22	160	65	1081

Results of applying the SVM on the testing data set without and with data normalization are summarised in Table 6.3, and Table. 6.4. The data normalization process much improved the accuracy. Without data normalization, the accuracy of predicting normal device is **90.27%**, device with response fault is **85.12%**, device with timing fault is **55.37%**, device with omission fault is **25.92%**, device with timing and omission fault is **37.96%**, and device with all faults is **20.83%**. After the data normalization, the accuracy of predicting normal device is **97.04%**, device with response fault is **96.90%**, device with timing fault is **87.62%**, device with omission fault is **71.61%**, device with timing and omission fault is **81.31%**, and device with all faults is **81.28%**. The SVM made wrong prediction for omission fault because it is not a linear distribution.

Artificial Neural Network

Table 6.5: Artificial Neural Network Without Normalization Confusion Matrix

Ground Truth	Normal	Response Fault	Timing Fault	Omission Fault	Combined of Timing and Omission	All Fault Combined
Normal	9143	11	2	18	3	0
Response Fault	8	1050	0	7	0	0
Timing Fault	2	0	2100	5	137	17
Omission Fault	15	15	26	1820	115	137
Combined of Timing and Omission	1	0	163	115	1911	204
All Fault Combined	0	0	2	149	341	837

Table 6.6: Artificial Neural Network With Normalization Confusion Matrix

Ground Truth	Normal	Response Fault	Timing Fault	Omission Fault	Combined of Timing and Omission	All Fault Combined
Normal	9165	2	0	7	0	3
Response Fault	3	1061	0	0	0	0
Timing Fault	5	0	2149	0	83	23
Omission Fault	27	7	4	1926	132	32
Combined of Timing and Omission	7	1	31	34	2212	110
All Fault Combined	0	1	1	62	27	1238

The ANN is configured to have one hidden layer with 100 nodes, the **sigmoid** activation function, and the **mean square error** loss function. Results of applying ANN on the testing data set without and with data normalization are summarised in Table 6.5, and Table. 6.6. The data normalization process slightly improved the accuracy. Without data normalization, the accuracy of predicting normal device is **99.63%**, device with response fault is **98.68%**, device with timing fault is **92.88%**, device with omission fault is **85.53%**, device with timing and omission fault is **80.43%**, and device with all faults is **62.93%**. After the data normalization, the accuracy of predicting normal device is **99.87%**, device with response fault is **99.72%**, device with timing fault is **95.05%**, device with omission fault is **85.18%**, device with timing and omission fault is **93.1%**, and device with all faults is **93.1%**. The ANN made wrong prediction for omission fault because it is similar to a fault that combines both the omission fault and the timing fault.

6.3 Summary

Three ML methods include Decision Tree (DT), Support Vector Machine, and Artificial Neural Network (ANN) are investigated for the experiments. Performances of those method with and without data normalization are summarized in Table 6.2, Table 6.4, Table 6.6, Table 6.1, Table 6.3, and Table 6.5. All three models achieve high accuracy in classifying normal devices and devices with response fault from the rest of faulty devices. However, the accuracy of detecting omission fault devices and devices with three errors combined is low. The ANN achieves the best performance (average accuracy 96.72%) with data normalization. The DT achieves the best performance (average accuracy 93.23%) without data normalization.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This dissertation is motivated by the problem that there is no data set to build AI integrated network management solutions for the IoT Area Network. In the scope of this dissertation, (i) a network traffic generation framework can generate traffic data of smart home networks, and (ii) implementations of ML techniques to verify the usability and reliability of the generated traffic dataset have been introduced. The home network simulator which is a main component of the proposed framework is configurable in order to create various types of network traffic for data set generation purposes. The summarized of this dissertation is as follows:

- Home Gateway (HGW) plays a vital role in the IoT area network, which is a central point to manage and represent for devices in the network. Thus, network traffics between HGW and devices is used to diagnose the health of devices in the network. In chapter 3, concepts and implementations of the HGW have been introduced. The HGW covers all requirements stated in the ITU-T Y.2070, and ITU-T Y.4113 includes operation, management, and integration with service platforms outside of the home network. The implementation of the ECHONET Lite HGW has been introduced. Two HGWs include ECHONET Lite-universAAL (*elite4u*), and ECHONET Lite-oneM2M (*eIPE*) have been deployed and tested. The *eIPE* can facilitate a network of ECHONET Lite devices into the oneM2M ecosystem, which is fundamental to build the Service Platform for smart homes. The *elite4u* integrated networks of ECHONET Lite devices into an ambient assisted living (AAL) platform, which enables AAL services for smart homes. The elite4u is the main component of the human-robot-environment interaction in the CARESSES project.

- The device emulator, which is proposed in Chapter 4, is the last part of the device-gateway interaction for network traffic generation. Since it is impossible to have a faulty commercial device to collect the abnormal network traffic, the device emulator is the answer to this problem. Firstly, the simulation of standard industrial devices is guarantee by the proposed device emulator. The experiment that involves simulated devices and real commercial devices was conducted. The result shows the same network traffic and behaviors of simulated and industrial devices. The faulty devices are implemented by extending a fault model for the distributed system. Totally four types of fault include crash fault, omission fault, timing fault, and response fault are simulated. By utilizing the docker platform, the device emulator achieved the automatic and scalable deployment. The memory usage **100MB** and CPU usage (**0.15%**) for a node are suitable to deploy on a large scale.

- Finally, in chapter 5 and chapter 6, the evaluation of network traffic generated by deploying the proposed simulator has been done. The network traffic in the data set that is in the form of captured packets is aggregated into bidirectional flows to reflect the device-gateway interactions by the proposed *Flowcal*. The network flow calculator, namely *Flowcal*, which is customizing for the IoT Area Network, supports the appointment of flow direction initiator and multicast flows. Three ML methods include decision tree (DT), support vector machine (SVM), and artificial neural network (ANN) have been investigated. The ANN achieves the best performance (average accuracy 96.72%) with data normalization, and the DT achieves the best performance (average accuracy 93.23%) in predicting device faults based on network traffic. without data normalization.

7.2 Future Work

To use AI integrated solution for network management, a high-performance computer to deploy the application is required. Since it is not realistic to have a high power computer in the home network, the integration with high reliability, availability, and scalability platform that support a big data analytics framework such as PNDA [1] is desired as future work. The mechanism to re-train the model with online data collection is also extensible as future work.

[1]http://pnda.io/

Bibliography

[1] A. H. Maslow. A theory of human motivation. *Psychological Review*, 50(4):370–396, July 1943.

[2] Mukhtiar Memon, Stefan Rahr Wagner, Christian Fischer Pedersen, Femina Hassan Aysha Beevi, and Finn Overgaard Hansen. Ambient assisted living healthcare frameworks, platforms, standards, and quality attributes. In *Sensors*, 2014.

[3] Cisco visual networking index: Forecast and trends, 2017–2022 white paper. Technical report, Cisco, 02 2019.

[4] L. Andersson, H. van Helvoort, R. Bonica, D. Romascanu, and S. Mansfield. Guidelines for the use of the "oam" acronym in the ietf. BCP 161, RFC Editor, June 2011.

[5] T. Mizrahi, N. Sprecher, E. Bellagamba, and Y. Weingarten. An overview of operations, administration, and maintenance (oam) tools. RFC 7276, RFC Editor, June 2014.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[7] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[8] International Telecommunication Union. *SERIES J: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS: Architecture of MediaHomeNet*, 07 2007.

[9] Network management system : Best practices white paper. Cisco, 2018.

[10] Annie Ibrahim Rana and Brendan Jennings. Semantic aware processing of user defined inference rules to manage home networks. *Journal of Network and Computer Applications*, 79:68–87, February 2017.

[11] W. De Donato, A. Pescapé, and A. Dainotti. Traffic identification engine: an open platform for traffic classification. *IEEE Network*, 28(2):56–64, March 2014.

[12] Raouf Boutaba, Mohammad A. Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M. Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16, Jun 2018.

[13] Y. Roh, G. Heo, and S. E. Whang. A survey on data collection for machine learning: A big data - ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2019.

[14] Krzysztof Grochla and Leszek Naruszewicz. Testing and scalability analysis of network management systems using device emulation. In Andrzej Kwiecie'n, Piotr Gaj, and Piotr Stera, editors, *Computer Networks*, pages 91–100, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[15] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang. Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 32(2):92–99, March 2018.

[16] A. Eswaradass, Xian-He Sun, and Ming Wu. Network bandwidth predictor (nbp): a system for online network performance forecasting. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, volume 1, pages 4 pp.–268, May 2006.

[17] P. Cortez, M. Rio, M. Rocha, and P. Sousa. Internet traffic forecasting using neural networks. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 2635–2642, July 2006.

[18] P. Bermolen and D. Rossi. Support vector regression for link load prediction. In *2008 4th International Telecommunication Networking Workshop on QoS in Multiservice IP Networks*, pages 268–273, Feb 2008.

[19] Samira Chabaa, Abdelouhab Zeroual, and Jilali Antari. Identification and prediction of internet traffic using artificial neural networks. *Journal of Intelligent Learning Systems and Applications*, 02(03):147–155, 2010.

[20] Yan Zhu, Guanghua Zhang, and Jing Qiu. Network traffic prediction based on particle swarm bp neural network. *JNW*, 8:2685–2691, 2013.

[21] Y. Li, H. Liu, W. Yang, D. Hu, and W. Xu. Inter-data-center network traffic prediction with elephant flows. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 206–213, April 2016.

[22] Zhitang Chen, Jiayao Wen, and Yanhui Geng. Predicting future traffic using hidden markov models. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pages 1–6, Nov 2016.

[23] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Yanhui Geng, Li Chen, K. Chen, and Hao Jin. Online flow size prediction for improved network routing. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pages 1–6, Nov 2016.

[24] C. S. Hood and C. Ji. Proactive network-fault detection [telecommunications]. *IEEE Transactions on Reliability*, 46(3):333–341, Sep. 1997.

[25] Okuthe P. Kogeda and Johnson I. Agbinya. Prediction of faults in cellular networks using bayesian network model. 2007.

[26] A. Snow, P. Rastogi, and G. Weckman. Assessing dependability of wireless networks using neural networks. In *MILCOM 2005 - 2005 IEEE Military Communications Conference*, pages 2809–2815 Vol. 5, Oct 2005.

[27] Yong Wang, Margaret Martonosi, and Li-Shiuan Peh. Predicting link quality using supervised learning in wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 11(3):71–83, July 2007.

[28] Xu Lu, Huiqiang Wang, Renjie Zhou, and Baoyu Ge. Using hessian locally linear embedding for autonomic failure prediction. In *2009 World Congress on Nature Biologically Inspired Computing (NaBIC)*, pages 772–776, Dec 2009.

[29] Zhilong Wang, Min Zhang, Danshi Wang, Chuang Song, Min Liu, Jin Li, Liqi Lou, and Zhuo Liu. Failure prediction using machine learning and time series in optical network. *Opt. Express*, 25(16):18553–18565, Aug 2017.

[30] S. Rao. Operational fault detection in cellular wireless base-stations. *IEEE Transactions on Network and Service Management*, 3(2):1–11, April 2006.

[31] Karwan Qader, Mo Adda, and Mouhammd Al-Kasassbeh. Comparative analysis of clustering techniques in network traffic faults classification. *International Journal of Innovative Research in Computer and Communication Engineering*, 5(4):6551–6563, 6 2017. DOI not working: 10.15680/IJIRCCE.2017.0504001.

[32] U. S. Hashmi, A. Darbandi, and A. Imran. Enabling proactive self-healing by data mining network failure logs. In *2017 International Conference on Computing, Networking and Communications (ICNC)*, pages 511–517, Jan 2017.

[33] H. Hajji. Statistical analysis of network traffic for adaptive faults detection. *IEEE Transactions on Neural Networks*, 16(5):1053–1063, Sep. 2005.

[34] Xu Lu, Huiqiang Wang, Renjie Zhou, and Baoyu Ge. Using hessian locally linear embedding for autonomic failure prediction. In *2009 World Congress on Nature Biologically Inspired Computing (NaBIC)*, pages 772–776, Dec 2009.

[35] The cost of security breaches. Technical report, Kaspersky Lab, 2015.

[36] A. L. Buczak and E. Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys Tutorials*, 18(2):1153–1176, Secondquarter 2016.

[37] S. Hettich and S. D. Bay. The uci kdd archive [http://kdd.ics.uci.edu], 1999.

[38] International Telecommunication Union. *Requirements and architecture of the home energy management system and home network services*, 01 2015.

[39] Home-network topology identifying protocol (htip). Technical report, Telecommunication Technology Committee, May 2017.

[40] Y. Mihara, T. Yamazaki, and A. Takehiro. Designing htip: Home network topology identifying protocol. In *2011 IEEE International Conference on Communications (ICC)*, pages 1–6, June 2011.

[41] Customer support functions for home network service platform. Technical report, Telecommunication Technology Committee, February 2016.

[42] International Telecommunication Union. *Requirements of the network for the Internet of things*, 09 2016.

[43] C. Pham, Y. Lim, and Y. Tan. Management architecture for heterogeneous iot devices in home network. In *2016 IEEE 5th Global Conference on Consumer Electronics*, pages 1–5, Oct 2016.

[44] Masaki Umejima. Japan's power meter deployment with echonet lite over ipv6. *New Breeze - Quarterly of the ITU Association of Japan*, 27(2):12–13, April 2015.

[45] ECHONET CONSORTIUM. *Detailed Requirements for ECHONET Device objects*, August 2017.

[46] Hiroyuki Fujita, Hiroshi Sugimura, Takashi Murakami, and Masao Isshiki. Development of echonet lite packet sending and receiving tool, ssng for iphone. Technical Report 14, Kanagawa Institute of Technology, may 2016.

[47] S. Saito, N. Ishikawa, and Y. Tsuchiya. Development of echonet lite-compliant home appliances control system using pucc protocols from smart devices. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 3, pages 200–204, July 2015.

[48] oneM2M. *oneM2M-TS 0023 Home Appliances Information Model and Mapping*, September 2017.

[49] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The semantic web: the roles of xml and rdf. *IEEE Internet Computing*, 4(5):63–73, Sept 2000.

[50] Laura Daniele, Frank den Hartog, and Jasper Roes. Study on semantic assets for smart appliances interoperability. Technical report, TNO, 2015.

[51] SmartM2M. Reference ontology and onem2m mapping. Technical Report ETSI TS 103 264 V1.1.1, ETSI, 11 2015.

[52] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5):907 – 928, 1995.

[53] Francisco Jose Garcia-Penalvo Juan Garcia and Roberto Theron. A survey on ontology metrics. In *Knowledge Management, Information Systems, E-Learning, and Sustainability Research*, pages 22–27. Springer Berlin Heidelberg, 2010.

[54] S. Staab and A. Maedche. Axioms are objects, too - ontology engineering beyond the modeling of concepts and relations. Technical report, 2000.

[55] ECHONET CONSORTIUM. *Part V ECHONET Lite System Design Guidelines*, May 2016.

[56] TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU. *Semantics based requirements and framework of the Internet of things*, February 2016.

[57] Mahdi Ben Alaya, Samir Medjiah, Thierry Monteil, and Khalil Drira. Towards Semantic Data Interoperability in oneM2M Standard. *IEEE Communications Magazine*, 53(12):pp. 35–41, December 2015.

[58] S. Sicari, A. Rizzardi, A. Coen-Porisini, L. A. Grieco, and T. Monteil. Secure om2m service platform. In *2015 IEEE International Conference on Autonomic Computing*, pages 313–318, July 2015.

[59] Sten Hanke, Christopher Mayer, Oliver Hoeftberger, Henriette Boos, Reiner Wichert, Mohammed-R. Tazari, Peter Wolf, and Francesco Furfari. universaal an open and consolidated aal platform. In *Ambient Assisted Living*, pages 127–140. Springer Berlin Heidelberg, 2011.

[60] Tom Zentek, Can Oliver Yumusak, Christian Reichelt, and Asarnusch Rashid. Which aal middleware matches my requirements? an analysis of current middleware systems and a framework for decision-support. In *Ambient Assisted Living*, pages 111–125. Springer International Publishing, 2015.

[61] Panou Maria, Cabrera Maria F., Bekiaris Evangelos, and Touliou Katerina. Ict services for prolonging independent living of the elderly with cognitive impairments, in life concept. *8th Forum Italiano dell'Ambient Assisted Living (ForitAAL)*, 217(Assistive Technology):6592013663, 2015.

[62] Laura Fiorini, Grazia D'Onofrio, Raffaele Limosani, Daniele Sancarlo, Antonio Greco, Francesco Giuliani, Antonio Kung, Paolo Dario, and Filippo Cavallo. Accra project: Agile co-creation for robots and aging. 2017.

[63] Diletta Romana Cacciagrano Rosario Culmone Luca Tesei Leonardo Vito Flavio Corradini, Emanuela Merelli. Activage: proactive and self-adaptive social sensor network for ageing people. In *The European Research Consortium for Informatics and Mathematics*, pages 36–37, 2011.

[64] Y. Lim, S. Y. Lim, M. Dat Nguyen, C. Li, and Y. Tan. Bridging between universaal and echonet for smart home environment. In *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 56–61, June 2017.

[65] V. C. Pham, Y. Lim, Y. Tan, and N. Y. Chong. Support for echonet-based smart home environments in the universaal ecosystem. In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–4, Jan 2018.

[66] Ha-Duong Bui, Cu Pham, Yuto Lim, Yasuo Tan, and Nak Young Chong. Integrating a humanoid robot into echonet-based smart home environments. In *Social Robotics*, pages 314–323, Cham, 2017. Springer International Publishing.

[67] Barbara Bruno, Nak Young Chong, Hiroko Kamide, Sanjeev Kanoria, Jaeryoung Lee, Yuto Lim, Amit Kumar Pandey, Chris Papadopoulos, Irena Papadopoulos, Federico Pecora, Alessandro Saffiotti, and Antonio Sgorbissa. The caresses eu-japan project: making assistive robots culturally competent, 2017.

[68] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible markup language (xml) 1.0, 2000.

[69] Thaha Muhammed and Riaz Shaikh. An analysis of fault detection strategies in wireless sensor networks. *Journal of Network and Computer Applications*, 78:267–287, 01 2017.

[70] Zainib Noshad, Nadeem Javaid, Tanzila Saba, Zahid Wadud, M.Q. Saleem, Mohammad Alzahrani, and Osama Sheta. Fault detection in wireless sensor networks through the random forest classifier. *Sensors*, 19:1568, 04 2019.

[71] Abhishek Sharma, Leana Golubchik, and Ramesh Govindan. Sensor faults detection methods and prevalence in real-world datasets. *TOSN*, 6, 01 2010.

[72] Tusher Chakraborty, Akshay Uttama Nambi, Ranveer Chandra, Rahul Sharma, Manohar Swaminathan, Zerina Kapetanovic, and Jonathan Appavoo. Fall-curve: A novel primitive for iot fault detection and isolation. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, SenSys '18, pages 95–107, New York, NY, USA, 2018. ACM.

[73] Flavin Cristian. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2):56–78, February 1991.

[74] Babak Bashari Rad, Harrison Bhatti, and Mohammad Ahmadi. An introduction to docker and analysis of its performance. *IJCSNS International Journal of Computer Science and Network Security*, 173:8, 03 2017.

[75] A. Helal, K. Cho, W. Lee, Y. Sung, J. W. Lee, and E. Kim. 3d modeling and simulation of human activities in smart spaces. In *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*, pages 112–119, Sep. 2012.

[76] J. Synnott, L. Chen, C. D. Nugent, and G. Moore. The creation of simulated activity datasets using a graphical intelligent environment simulation tool. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 4143–4146, Aug 2014.

[77] Nasser Alshammari, Talal Alshammari, Mohamed Sedky, Justin Champion, and Carolin Bauer. Openshs: Open smart home simulator. *Sensors*, 17(5), 2017.

[78] Hiroshi Nishikawa, Shinya Yamamoto, Morihiko Tamai, Kouji Nishigaki, Tomoya Kitani, Naoki Shibata, Keiichi Yasumoto, and Minoru Ito. Ubireal: Realistic smartspace simulator for systematic testing. In *UbiComp*, 2006.

[79] Jumphon Lertlakkhanakul, Sun-Hwie Hwang, and Jin-Won Choi. Avatar expression agent in virtual architecture. In Andy Dong, Andrew Vande Moere, and John S. Gero, editors, *Computer-Aided Architectural Design Futures (CAADFutures) 2007*, pages 361–372, Dordrecht, 2007. Springer Netherlands.

[80] Salman Azhar. Building information modeling (bim): Trends, benefits, risks, and challenges for the aec industry. *Leadership and Management in Engineering*, 11:241–252, 07 2011.

[81] D. Harrington, R. Presuhn, and B. Wijnen. An architecture for describing simple network management protocol (snmp) management frameworks. STD 62, RFC Editor, December 2002. http://www.rfc-editor.org/rfc/rfc3411.txt.

[82] B. Claise. Cisco systems netflow services export version 9. RFC 3954, RFC Editor, October 2004. http://www.rfc-editor.org/rfc/rfc3954.txt.

[83] B. Claise. Specification of the ip flow information export (ipfix) protocol for the exchange of ip traffic flow information. RFC 5101, RFC Editor, January 2008. http://www.rfc-editor.org/rfc/rfc5101.txt.

[84] Ludovic Noirie, Emmanuel Dotaro, Giovanna Carofiglio, Arnaud Dupas, Pascal Pecci, Daniel Popa, and Georg Post. Semantic networking: Flow-based, traffic-aware, and self-managed networking. *Bell Labs Technical Journal*, 14(2):23–38, August 2009.

[85] J. Park, H. Tyan, and C. . J. Kuo. Internet traffic classification for scalable qos provision. In *2006 IEEE International Conference on Multimedia and Expo*, pages 1221–1224, July 2006.

[86] B. Trammell and E. Boschi. Bidirectional flow export using ip flow information export (ipfix). RFC 5103, RFC Editor, January 2008. http://www.rfc-editor.org/rfc/rfc5103.txt.

[87] P. Minarik, J. Vykopal, and V. Krmicek. Improving host profiling with bidirectional flows. In *2009 International Conference on Computational Science and Engineering*, volume 3, pages 231–237, Aug 2009.

[88] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of tor traffic using time based features. In *ICISSP*, pages 253–262, 2017.

[89] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pages 407–414, 2016.

[90] Laya Taheri, Andi Fitriah Abdul Kadir, and Arash Habibi Lashkari. Extensible android malware detection and family classification using network-flows and api-calls. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–8. IEEE, 2019.

[91] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani. Developing realistic distributed denial of service (ddos) attack dataset and taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–8. IEEE, 2019.

[92] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116, 2018.

[93] Matej Slapnik, Darja Istenič, Marina Pintar, and Andrej Udovč. Extending life cycle assessment normalization factors and use of machine learning – a slovenian case study. *Ecological Indicators*, 50:161–172, March 2015.

[94] Hervé Abdi and Lynne J. Williams. Principal component analysis. *WIREs Comput. Stat.*, 2(4):433–459, July 2010.

[95] Chandrika Kamath. *Scientific Data Mining.* Society for Industrial and Applied Mathematics, January 2009.

[96] José Marcio Luna, Efstathios D. Gennatas, Lyle H. Ungar, Eric Eaton, Eric S. Diffenderfer, Shane T. Jensen, Charles B. Simone, Jerome H. Friedman, Timothy D. Solberg, and Gilmer Valdes. Building more accurate decision trees with the additive tree. *Proceedings of the National Academy of Sciences*, 116(40):19887–19893, 2019.

[97] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

[98] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

[99] Marti A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, July 1998.

[100] Gidudu Anthony, Hulley Gregg, and Marwala Tshilidzi. Image classification using svms: One-against-one vs one-against-all, 2007.

[101] S. Mu, C. Yin, and S. Tian. A novel all-at-once learning method for multi-class support vector machine. In *2010 3rd International Congress on Image and Signal Processing*, volume 4, pages 1543–1546, Oct 2010.

[102] Wiesław Chmielnicki and Katarzyna Stapor. A new approach to multi-class svm-based classification using error correcting output codes. In Robert Burduk, Marek Kurzyński, Michał Woźniak, and Andrzej Żołnierek, editors, *Computer Recognition Systems 4*, pages 499–506, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[103] L Auria and R Moro. Advantages and disadvantages of support vector machines. *Credit Risk Assessment Revisited: Methodological Issues and Practical Implications*, pages 49–68, 2007.

[104] Kevin L. Priddy and Paul E. Keller. *Artificial Neural Networks: An Introduction (SPIE Tutorial Texts in Optical Engineering, Vol. TT68).* SPIE- International Society for Optical Engineering, 2005.

Publications and Awards

Journals

[1] Van Cu Pham, Yoshiki Makino, Khoa Pho, Yuto Lim, and Yasuo Tan: IoT Area Network Simulator For Network Dataset Generation, Special issue of Ubiquitous Computing Systems (IX), *Journal of Information Processing.* (**Submitted, Under Review**)

[2] Van Cu Pham, Yuto Lim, Antonio Sgorbissa,and Yasuo Tan: An Ontology-driven ECHONET Lite Adaptation Layer for Smart Homes, *Journal of Information Processing*,vol 27, pp360-368, May 2019. https://doi.org/10.2197/ipsjjip.27.360.

International Conference papers

[3] Van Cu Pham, Yuto Lim, Ha Duong Bui, Yasuo Tan, Nak Young Chong and Antonio Sgorbissa: An Experimental Study on Culturally Competent Robot for Smart Home Environment, *The 34th International Conference on Advanced Information Networking and Applications (AINA).*(**Accepted, to be published**)

[4] Van Cu Pham, Yuto Lim, and Yasuo Tan: An onem2m interworking proxy entity for echonet lite protocol, *2019 IEEE 8th Global Conference on Consumer Electronics*, pp1-5, Oct 2019.

[5] Van Cu Pham, Yoshiki Makino, Yuto Lim, and Yasuo Tan: Semantic service gateway for echonet based smart homes, *2019 22nd Conference on Innovation in Clouds,Internet and Networks and Workshops (ICIN)*, pp175-179, Feb 2019. https://doi.org/10.1109/ICIN.2019.8685883.

[6] Van Cu Pham, Yuto Lim, and Yasuo Tan: A platform for integrating alexa voice service into echonet-based smart homes, *2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, pp1-5, May 2018. https://doi.org/10.1109/ICCE-China.2018.8448893.

[7] Van Cu Pham, Yuto Lim, Yasuo Tan, and Nak Young Chong: Support for echonet-based smart home environments in the universaal ecosystem, *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pp1-4, Jan 2018. https://doi.org/10.1109/ICCE.2018.8326218.

[8] Ha-Duong Bui, Van Cu Pham, Yuto Lim, Yasuo Tan, and Nak Young Chong: Integrating a humanoid robot into echonet-based smart home environments, *9th International Conference on Social Robotics*, pp314-323, 2017.

[9] Van Cu Pham, Tan Le, Yuto Lim, and Yasuo Tan: An architecture for supporting ras on linux-based iot gateways, *2017 IEEE 6th Global Conference on Consumer Electronics*, pp1-5, Oct 2017. https://doi.org/10.1109/GCCE.2017.8229234.

[10] Van Cu Pham, Yuto Lim, and Yasuo Tan: Management architecture for heterogeneous IoT devices in home network, *2016 IEEE 5th Global Conference on Consumer Electronics*, pp1-5, Oct 2016. https://doi.org/10.1109/GCCE.2016.7800448.

Japan Domestic Conference papers

[11] Van Cu Pham, Yoshiki Makino, and Yasuo Tan: Support for ECHONET Lite Protocol in the oneM2M Ecosystem, *IEICE General Conference 2019*, Mar 2019.

[12] Van Cu Pham, Yuto Lim, and Yasuo Tan: Integrating Alexa Voice Service Into ECHONET-based Home Networks, *IEICE General Conference 2018*, Mar 2018.

[13] Van Cu Pham, Yuto Lim, and Yasuo Tan: Cloud-based Solution for Connecting Multiple Home Networks using universAAL Space Gateway, *IEICE Society Conference 2017*, Sep 2017.

[14] Van Cu Pham, Yuto Lim, and Yasuo Tan: Management Architecture for Heterogeneous IoT Devices in Home Network, *Joint Conference of Hokuriku Chapters of Electrical Societies 2016 (JHES)*, Sep 2016.

Workshops

[15] Van Cu Pham, Yoshiki Makino, and Yasuo Tan: Support for ECHONET Lite Protocol in the oneM2M Ecosystem, *oneM2M Industry Day Kanazawa*, Dec 2018.

[16] Van Cu Pham, Yuto Lim, and Yasuo Tan: Integration of ECHONET Lite Protocol into The universAAL Platform, *JAIST World Conference 2018*, Feb 2018.

Awards

- Best English Paper Award in 2017, IEICE Technical Committee on Information and Communication Management (ICM), March 2018, Okinawa, Japan.
- Best Poster Award, JAIST World Conference 2018, February 2018.

Appendices

Appendix A: XML Device Object Configuration

Listing 7.1: **Device Object Configuration of A Toshiba LEDD85021N-LS**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<device>
  <profile>
    <property epc="80">
      <const>31</const>
    </property>
    <property epc="82">
      <const>010a0100</const>
    </property>
    <property epc="83">
      <const>fe00001bb86b233e44ad00000000000000</const>
    </property>
    <property epc="8A">
      <const>00001b</const>
    </property>
    <property epc="8D">
      <const>6238366232333365343461 64</const>
    </property>
    <property epc="9D">
      <const>01d5</const>
    </property>
  <property epc="9E">
      <const>0280bf</const>
    </property>
  <property epc="9F">
      <const>0e8082838a8d9d9e9fbfd3d4d6d7f0</const>
    </property>
    <property epc="BF">
      <const>8001</const>
    </property>
    <property epc="D3">
      <const>000001</const>
    </property>
    <property epc="D4">
      <const>0002</const>
    </property>
    <property epc="D5">
      <const>01029001</const>
    </property>
    <property epc="D6">
      <const>01029001</const>
    </property>
    <property epc="D7">
      <const>010290</const>
    </property>
    <property epc="F0">
      <const>42</const>
    </property>
  </profile>
  <object ceoj="0290">
    <property epc="80" get="enabled" notify="enabled" set="enabled">
      <file>
        <value default="31">192.168.2.90/029001/0x80</value>
        <notify>192.168.2.90/029001/0x80notify</notify>
        <block>192.168.2.90/029001/0x80block</block>
      </file>
    </property>
    <property epc="81" get="enabled" notify="enabled" set="enabled">
      <file>
        <value default="08">192.168.2.90/029001/0x81</value>
        <notify>192.168.2.90/029001/0x81notify</notify>
        <block>192.168.2.90/029001/0x81block</block>
      </file>
    </property>
    <property epc="82" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="00004200">192.168.2.90/029001/0x82</value>
        <block>192.168.2.90/029001/0x82block</block>
      </file>
    </property>
    <property epc="86" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="0200001b0000">192.168.2.90/029001/0x86</value>
        <block>192.168.2.90/029001/0x86block</block>
      </file>
    </property>
    <property epc="88" get="enabled" notify="enabled" set="disabled">
      <file>
        <value default="42">192.168.2.90/029001/0x88</value>
        <notify>192.168.2.90/029001/0x88notify</notify>
        <block>192.168.2.90/029001/0x88block</block>
      </file>
    </property>
    <property epc="89" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="0000">192.168.2.90/029001/0x89</value>
        <block>192.168.2.90/029001/0x89block</block>
      </file>
    </property>
    <property epc="8A" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="00001b">192.168.2.90/029001/0x8A</value>
        <block>192.168.2.90/029001/0x8Ablock</block>
      </file>
    </property>
    <property epc="8B" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="000001">192.168.2.90/029001/0x8B</value>
        <block>192.168.2.90/029001/0x8Bblock</block>
      </file>
    </property>
    <property epc="8C" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="4c4544442d4c543120202020">192.168.2.90/029001/0x8C</value>
        <block>192.168.2.90/029001/0x8Cblock</block>
      </file>
    </property>
    <property epc="8D" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="303030303030303030303030">192.168.2.90/029001/0x8D</value>
        <block>192.168.2.90/029001/0x8Dblock</block>
      </file>
    </property>
    <property epc="8E" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="07dd0511">192.168.2.90/029001/0x8E</value>
        <block>192.168.2.90/029001/0x8Eblock</block>
      </file>
    </property>
    <property epc="8F" get="enabled" notify="disabled" set="enabled">
      <file>
        <value default="42">192.168.2.90/029001/0x8F</value>
        <block>192.168.2.90/029001/0x8Fblock</block>
      </file>
    </property>
    <property epc="9D" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="05808188feff">192.168.2.90/029001/0x9D</value>
        <block>192.168.2.90/029001/0x9Dblock</block>
      </file>
    </property>
    <property epc="9E" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="0a80818ff4f6f7f8fdfeff">192.168.2.90/029001/0x9E</value>
        <block>192.168.2.90/029001/0x9Eblock</block>
      </file>
    </property>
    <property epc="9F" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="1801018100880081808101010101838383">192.168.2.90/029001/0x9F</value>
        <block>192.168.2.90/029001/0x9Fblock</block>
      </file>
    </property>
    <property epc="B4" get="enabled" notify="disabled" set="disabled">
      <file>
        <value default="6400">192.168.2.90/029001/0xB4</value>
        <block>192.168.2.90/029001/0xB4block</block>
      </file>
    </property>
    <property epc="F2" get="enabled" notify="disabled" set="disabled">
      <file>
```

```
        <value default="640042">192.168.2.90/029001/0xF2</value>
        <block>192.168.2.90/029001/0xF2block</block>
      </file>
    </property>
    <property epc="F4" get="enabled" notify="disabled" set="enabled">
      <file>
        <value default="30">192.168.2.90/029001/0xF4</value>
        <block>192.168.2.90/029001/0xF4block</block>
      </file>
    </property>
    <property epc="F6" get="enabled" notify="disabled" set="enabled">
      <file>
        <value default="42">192.168.2.90/029001/0xF6</value>
        <block>192.168.2.90/029001/0xF6block</block>
      </file>
    </property>
     <property epc="F7" get="enabled" notify="disabled" set="enabled">
      <file>
        <value default="00">192.168.2.90/029001/0xF7</value>
        <block>192.168.2.90/029001/0xF7block</block>
      </file>
    </property>
    <property epc="F8" get="enabled" notify="disabled" set="enabled">
      <file>
        <value default="00">192.168.2.90/029001/0xF8</value>
        <block>192.168.2.90/029001/0xF8block</block>
      </file>
    </property>
    <property epc="FD" get="enabled" notify="disabled" set="enabled">
      <file>
        <value default="140008">192.168.2.90/029001/0xFD</value>
        <block>192.168.2.90/029001/0xFDblock</block>
      </file>
    </property>
    <property epc="FE" get="enabled" notify="enabled" set="enabled">
      <file>
        <value default="30">192.168.2.90/029001/0xFE</value>
        <notify>192.168.2.90/029001/0xFEnotify</notify>
        <block>192.168.2.90/029001/0xFEblock</block>
      </file>
    </property>
    <property epc="FF" get="enabled" notify="enabled" set="enabled">
      <file>
        <value default="31">192.168.2.90/029001/0xFF</value>
        <notify>192.168.2.90/029001/0xFFnotify</notify>
        <block>192.168.2.90/029001/0xFFblock</block>
      </file>
    </property>
  </object>
</device>
```

Appendix B: Proposed HGW and CARESSES System

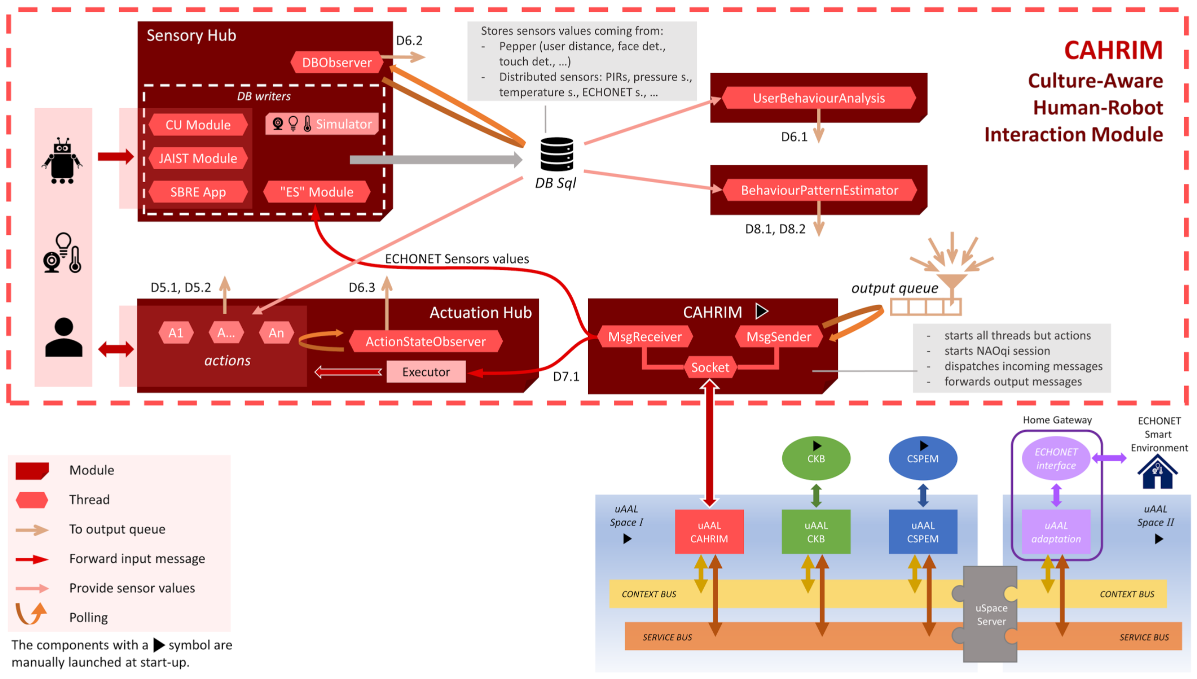


Figure 7.1: Proposed Home Gateway in the CARESSES Ecosystem

The proposed HGW implementation plays an important role for the robot-human-environment interaction of the CARESSES project. The HGW provides APIs for other application to interacts with devices of smart home via Local Area Network and cloud server.

Appendix C: Device Configuration Information

Table 7.1: Home Network Simulator Device Description

IPAddress	DeviceObject	ErrorDetail	ErrorClass
192.168.2.100	TemperatureSensor	NoError	Normal
192.168.2.100	HumiditySensor	NoError	Normal
192.168.2.100	IlluminanceSensor	NoError	Normal
192.168.2.101	TemperatureSensor	NoError	Normal
192.168.2.101	HumiditySensor	NoError	Normal
192.168.2.101	IlluminanceSensor	NoError	Normal
192.168.2.102	TemperatureSensor	NoError	Normal
192.168.2.102	HumiditySensor	NoError	Normal
192.168.2.102	IlluminanceSensor	NoError	Normal
192.168.2.103	TemperatureSensor	NoError	Normal
192.168.2.103	HumiditySensor	NoError	Normal
192.168.2.103	IlluminanceSensor	NoError	Normal
192.168.2.104	TemperatureSensor	NoError	Normal
192.168.2.104	HumiditySensor	NoError	Normal

192.168.2.104	IlluminanceSensor	NoError	Normal
192.168.2.105	TemperatureSensor	NoError	Normal
192.168.2.105	HumiditySensor	NoError	Normal
192.168.2.105	IlluminanceSensor	NoError	Normal
192.168.2.106	TemperatureSensor	NoError	Normal
192.168.2.106	HumiditySensor	NoError	Normal
192.168.2.106	IlluminanceSensor	NoError	Normal
192.168.2.107	TemperatureSensor	NoError	Normal
192.168.2.107	HumiditySensor	NoError	Normal
192.168.2.107	IlluminanceSensor	NoError	Normal
192.168.2.108	TemperatureSensor	NoError	Normal
192.168.2.108	HumiditySensor	NoError	Normal
192.168.2.108	IlluminanceSensor	NoError	Normal
192.168.2.109	TemperatureSensor	NoError	Normal
192.168.2.109	HumiditySensor	NoError	Normal
192.168.2.109	IlluminanceSensor	NoError	Normal
192.168.2.110	TemperatureSensor	NoError	Normal
192.168.2.110	HumiditySensor	NoError	Normal
192.168.2.110	IlluminanceSensor	NoError	Normal
192.168.2.111	HumanDetectionSensor	NoError	Normal
192.168.2.111	HumanDetectionSensor	NoError	Normal
192.168.2.112	HumanDetectionSensor	NoError	Normal
192.168.2.112	HumanDetectionSensor	NoError	Normal
192.168.2.112	HumanDetectionSensor	NoError	Normal
192.168.2.113	HumanDetectionSensor	NoError	Normal
192.168.2.113	HumanDetectionSensor	NoError	Normal
192.168.2.114	HumanDetectionSensor	NoError	Normal
192.168.2.114	HumanDetectionSensor	NoError	Normal
192.168.2.115	HumanDetectionSensor	NoError	Normal
192.168.2.115	HumanDetectionSensor	NoError	Normal
192.168.2.116	HumanDetectionSensor	NoError	Normal
192.168.2.116	HumanDetectionSensor	NoError	Normal
192.168.2.117	HumanDetectionSensor	NoError	Normal
192.168.2.117	HumanDetectionSensor	NoError	Normal
192.168.2.118	HumanDetectionSensor	NoError	Normal
192.168.2.118	HumanDetectionSensor	NoError	Normal
192.168.2.119	HumanDetectionSensor	NoError	Normal
192.168.2.119	HumanDetectionSensor	NoError	Normal
192.168.2.120	HumanDetectionSensor	NoError	Normal
192.168.2.120	HumanDetectionSensor	NoError	Normal
192.168.2.121	HumanDetectionSensor	NoError	Normal
192.168.2.121	HumanDetectionSensor	NoError	Normal
192.168.2.122	HumanDetectionSensor	NoError	Normal
192.168.2.122	HumanDetectionSensor	NoError	Normal
192.168.2.123	Lighting	NoError	Normal
192.168.2.123	Lighting	NoError	Normal
192.168.2.124	Lighting	NoError	Normal
192.168.2.124	Lighting	NoError	Normal

192.168.2.125	Lighting	NoError	Normal
192.168.2.125	Lighting	NoError	Normal
192.168.2.126	Lighting	NoError	Normal
192.168.2.126	Lighting	NoError	Normal
192.168.2.127	Lighting	NoError	Normal
192.168.2.127	Lighting	NoError	Normal
192.168.2.128	Lighting	NoError	Normal
192.168.2.128	Lighting	NoError	Normal
192.168.2.129	Lighting	NoError	Normal
192.168.2.129	Lighting	NoError	Normal
192.168.2.130	Lighting	NoError	Normal
192.168.2.131	Lighting	NoError	Normal
192.168.2.131	Lighting	NoError	Normal
192.168.2.132	Lighting	NoError	Normal
192.168.2.132	Lighting	NoError	Normal
192.168.2.143	WaterFlowRateSensor	NoError	Normal
192.168.2.143	WaterFlowRateSensor	NoError	Normal
192.168.2.144	WaterFlowRateSensor	NoError	Normal
192.168.2.144	WaterFlowRateSensor	NoError	Normal
192.168.2.145	TemperatureSensor	NoError	Normal
192.168.2.145	HumiditySensor	NoError	Normal
192.168.2.145	AirSpeedSensor	NoError	Normal
192.168.2.147	InterCom	NoError	Normal
192.168.2.147	FireSensor	NoError	Normal
192.168.2.148	DoorLock	NoError	Normal
192.168.2.151	OpenCloseSensor	NoError	Normal
192.168.2.151	OpenCloseSensor	NoError	Normal
192.168.2.152	OpenCloseSensor	NoError	Normal
192.168.2.152	OpenCloseSensor	NoError	Normal
192.168.2.153	OpenCloseSensor	NoError	Normal
192.168.2.153	OpenCloseSensor	NoError	Normal
192.168.2.154	OpenCloseSensor	NoError	Normal
192.168.2.154	OpenCloseSensor	NoError	Normal
192.168.2.155	OpenCloseSensor	NoError	Normal
192.168.2.156	OpenCloseSensor	NoError	Normal
192.168.2.156	OpenCloseSensor	NoError	Normal
192.168.2.157	ElectricCurtain	NoError	Normal
192.168.2.158	ElectricCurtain	NoError	Normal
192.168.2.159	ElectricWindow	NoError	Normal
192.168.2.160	ElectricWindow	NoError	Normal
192.168.2.161	ElectricWindow	NoError	Normal
192.168.2.163	ElectricWindow	NoError	Normal
192.168.2.164	ElectricWindow	NoError	Normal
192.168.2.166	ElectricWindow	NoError	Normal
192.168.2.167	ElectricWindow	NoError	Normal
192.168.2.168	ElectricWindow	NoError	Normal
192.168.2.170	AirConditioner	NoError	Normal
192.168.2.171	AirConditioner	NoError	Normal

192.168.2.172	AirConditioner	NoError	Normal
192.168.2.173	AirConditioner	NoError	Normal
192.168.2.174	AirConditioner	NoError	Normal
192.168.2.175	AirConditioner	NoError	Normal
192.168.2.183	ElectricCurtain	NoError	Normal
192.168.2.184	ElectricCurtain	NoError	Normal
192.168.2.200	Refrigerator	NoError	Normal
192.168.2.201	HotWaterPot	NoError	Normal
192.168.2.203	Stove	NoError	Normal
192.168.2.204	RiceCooker	NoError	Normal
192.168.2.205	TV	NoError	Normal
192.168.2.206	TV	NoError	Normal
192.168.2.207	Radio	NoError	Normal
192.168.2.99	TemperatureSensor	Delay:500	Delay
192.168.2.99	HumiditySensor	Delay:500	Delay
192.168.2.99	IlluminanceSensor	Delay:500	Delay
192.168.2.98	TemperatureSensor	Delay:1000	Delay
192.168.2.98	HumiditySensor	Delay:1000	Delay
192.168.2.98	IlluminanceSensor	Delay:1000	Delay
192.168.2.97	TemperatureSensor	Delay:2000	Delay
192.168.2.97	HumiditySensor	Delay:2000	Delay
192.168.2.97	IlluminanceSensor	Delay:2000	Delay
192.168.2.96	TemperatureSensor	MissingData	MissingData
192.168.2.96	HumiditySensor	MissingData	MissingData
192.168.2.96	IlluminanceSensor	MissingData	MissingData
192.168.2.95	TemperatureSensor	Drop:10	PacketDrop
192.168.2.95	HumiditySensor	Drop:10	PacketDrop
192.168.2.95	IlluminanceSensor	Drop:10	PacketDrop
192.168.2.94	TemperatureSensor	Drop:50	PacketDrop
192.168.2.94	HumiditySensor	Drop:50	PacketDrop
192.168.2.94	IlluminanceSensor	Drop:50	PacketDrop
192.168.2.93	TemperatureSensor	Drop:80	PacketDrop
192.168.2.93	HumiditySensor	Drop:80	PacketDrop
192.168.2.93	IlluminanceSensor	Drop:80	PacketDrop
192.168.2.92	TemperatureSensor	Drop:30—Delay:1500	Delay&PacketDrop
192.168.2.92	HumiditySensor	Drop:30—Delay:1500	Delay&PacketDrop
192.168.2.92	IlluminanceSensor	Drop:30—Delay:1500	Delay&PacketDrop
192.168.2.91	TemperatureSensor	Drop:60—Delay:200	Delay&PacketDrop
192.168.2.91	HumiditySensor	Drop:60—Delay:200	Delay&PacketDrop
192.168.2.91	IlluminanceSensor	Drop:60—Delay:200	Delay&PacketDrop
192.168.2.90	TemperatureSensor	Drop:90—Delay:800	Delay&PacketDrop
192.168.2.90	HumiditySensor	Drop:90—Delay:800	Delay&PacketDrop
192.168.2.90	IlluminanceSensor	Drop:90—Delay:800	Delay&PacketDrop
192.168.2.89	TemperatureSensor	Drop:10—Delay:900—MissingData	AllErrorCombined
192.168.2.89	HumiditySensor	Drop:10—Delay:900—MissingData	AllErrorCombined
192.168.2.89	IlluminanceSensor	Drop:10—Delay:900—MissingData	AllErrorCombined
192.168.2.88	HumanDetectionSensor	Delay:300	Delay
192.168.2.88	HumanDetectionSensor	Delay:300	Delay

192.168.2.87	HumanDetectionSensor	Delay:800	Delay
192.168.2.87	HumanDetectionSensor	Delay:800	Delay
192.168.2.87	HumanDetectionSensor	Delay:800	Delay
192.168.2.86	HumanDetectionSensor	Delay:1800	Delay
192.168.2.86	HumanDetectionSensor	Delay:1800	Delay
192.168.2.85	HumanDetectionSensor	Delay:3800	Delay
192.168.2.85	HumanDetectionSensor	Delay:3800	Delay
192.168.2.84	HumanDetectionSensor	MissingData	MissingData
192.168.2.84	HumanDetectionSensor	MissingData	MissingData
192.168.2.83	HumanDetectionSensor	Drop:20	PacketDrop
192.168.2.83	HumanDetectionSensor	Drop:20	PacketDrop
192.168.2.82	HumanDetectionSensor	Drop:60	PacketDrop
192.168.2.82	HumanDetectionSensor	Drop:60	PacketDrop
192.168.2.81	HumanDetectionSensor	Drop:40—Delay:1000	Delay&PacketDrop
192.168.2.81	HumanDetectionSensor	Drop:40—Delay:1000	Delay&PacketDrop
192.168.2.80	HumanDetectionSensor	Drop:80—Delay:400	Delay&PacketDrop
192.168.2.80	HumanDetectionSensor	Drop:80—Delay:400	Delay&PacketDrop
192.168.2.79	HumanDetectionSensor	Drop:10—Delay:3000	Delay&PacketDrop
192.168.2.79	HumanDetectionSensor	Drop:10—Delay:3000	Delay&PacketDrop
192.168.2.78	HumanDetectionSensor	Drop:30—Delay:1200—MissingData	AllErrorCombined
192.168.2.78	HumanDetectionSensor	Drop:30—Delay:1200—MissingData	AllErrorCombined
192.168.2.77	HumanDetectionSensor	Drop:50—Delay:500—MissingData	AllErrorCombined
192.168.2.77	HumanDetectionSensor	Drop:50—Delay:500—MissingData	AllErrorCombined
192.168.2.76	Lighting	Delay:100	Delay
192.168.2.76	Lighting	Delay:100	Delay
192.168.2.75	Lighting	Delay:1200	Delay
192.168.2.75	Lighting	Delay:1200	Delay
192.168.2.74	Lighting	MissingData	MissingData
192.168.2.74	Lighting	MissingData	MissingData
192.168.2.73	Lighting	Drop:80	PacketDrop
192.168.2.73	Lighting	Drop:80	PacketDrop
192.168.2.72	Lighting	Drop:15	PacketDrop
192.168.2.72	Lighting	Drop:15	PacketDrop
192.168.2.71	Lighting	Drop:20—Delay:3500	Delay&PacketDrop
192.168.2.71	Lighting	Drop:20—Delay:3500	Delay&PacketDrop
192.168.2.70	Lighting	Drop:70—Delay:3100	Delay&PacketDrop
192.168.2.70	Lighting	Drop:70—Delay:3100	Delay&PacketDrop
192.168.2.69	Lighting	Drop:55—Delay:1200	Delay&PacketDrop
192.168.2.68	Lighting	Drop:25—Delay:700—MissingData	AllErrorCombined
192.168.2.68	Lighting	Drop:25—Delay:700—MissingData	AllErrorCombined
192.168.2.67	Lighting	Drop:40—Delay:1700—MissingData	AllErrorCombined
192.168.2.67	Lighting	Drop:40—Delay:1700—MissingData	AllErrorCombined
192.168.2.66	WaterFlowRateSensor	Delay:200	Delay
192.168.2.66	WaterFlowRateSensor	Delay:200	Delay
192.168.2.65	WaterFlowRateSensor	DropRate:60	PacketDrop
192.168.2.65	WaterFlowRateSensor	DropRate:60	PacketDrop
192.168.2.64	TemperatureSensor	Drop:50—Delay:1900—MissingData	AllErrorCombined
192.168.2.64	HumiditySensor	Drop:50—Delay:1900—MissingData	AllErrorCombined

192.168.2.64	IlluminanceSensor	Drop:50—Delay:1900—MissingData	AllErrorCombined
192.168.2.63	InterCom	Droprate:85	PacketDrop
192.168.2.63	FireSensor	Droprate:85	PacketDrop
192.168.2.62	DoorLock	Delay:4000	Delay
192.168.2.61	OpenCloseSensor	MissingData	MissingData
192.168.2.61	OpenCloseSensor	MissingData	MissingData
192.168.2.60	OpenCloseSensor	Drop:5—Delay:1800	Delay&PacketDrop
192.168.2.60	OpenCloseSensor	Drop:5—Delay:1800	Delay&PacketDrop
192.168.2.59	OpenCloseSensor	Drop:35—Delay:800	Delay&PacketDrop
192.168.2.59	OpenCloseSensor	Drop:35—Delay:800	Delay&PacketDrop
192.168.2.58	OpenCloseSensor	Drop:45—Delay:50—MissingData	AllErrorCombined
192.168.2.58	OpenCloseSensor	Drop:45—Delay:50—MissingData	AllErrorCombined
192.168.2.57	OpenCloseSensor	Delay:1500	Delay
192.168.2.56	OpenCloseSensor	Drop:65	PacketDrop
192.168.2.56	OpenCloseSensor	Drop:65	PacketDrop
192.168.2.55	ElectricCurtain	Delay:4500	Delay
192.168.2.54	ElectricCurtain	Drop:75	PacketDrop
192.168.2.51	ElectricWindow	MissingData	MissingData
192.168.2.50	ElectricWindow	Delay:2500	Delay
192.168.2.49	ElectricWindow	Delay:4900	Delay
192.168.2.48	ElectricWindow	Drop:17—Delay:3100	Delay&PacketDrop
192.168.2.47	ElectricWindow	Drop:30—Delay:250	Delay&PacketDrop
192.168.2.46	ElectricWindow	Drop:50—Delay:2700	Delay&PacketDrop
192.168.2.45	ElectricWindow	Drop70—Delay:2700—MissingData	AllErrorCombined
192.168.2.44	ElectricWindow	Drop:25	PacketDrop
192.168.2.43	AirConditioner	MissingData	MissingData
192.168.2.42	AirConditioner	Drop:30	PacketDrop
192.168.2.41	AirConditioner	Drop:95	PacketDrop
192.168.2.40	AirConditioner	Delay:200	Delay
192.168.2.39	AirConditioner	Drop:10—Delay:4000	Delay&PacketDrop
192.168.2.38	AirConditioner	Drop:10—Delay:4000—MissingData	AllErrorCombined
192.168.2.53	ElectricCurtain	Drop:85—Delay:2200	Delay&PacketDrop
192.168.2.52	ElectricCurtain	MissingData	MissingData
192.168.2.37	Refrigerator	Drop:40	PacketDrop
192.168.2.36	HotWaterPot	Drop:60—Delay:800	Delay&PacketDrop
192.168.2.35	Stove	MissingData	MissingData
192.168.2.34	RiceCooker	Drop:60—Delay:800—MissingData	AllErrorCombined
192.168.2.33	TV	Delay:3000	Delay
192.168.2.32	TV	Drop:65	PacketDrop
192.168.2.31	Radio	Drop:65—Delay:3900	Delay&PacketDrop