Electronic Preprint for Journal of Information Processing Vol.28

Regular Paper

IoT Area Network Simulator For Network Dataset Generation

Van Cu Pham[1,a)] Yoshiki Makino[1,b)] Khoa Pho[1,c)] Yuto Lim[1,d)] Yasuo Tan[1,e)]

Received: December 23, 2019, Accepted: July 7, 2020

Abstract: With the development of information and communication technology (ICT), smart home technologies are expecting to be a potential solution for the population aging problem. 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 expand 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 the operational aspects of simulated devices and environments. The network communication aspects have not been fulfilled so far. Following the development of the Internet of Things, smart home networks are more complex, and users in the smart home are ordinary people without knowledge about network management. Therefore, **intelligent operation, administration, and maintenance** (OAM) services for smart homes are desired. This paper proposes a smart home network simulator to generate a network dataset of a home network, which is essential to develop machine learning technologies for providing intelligent OAM services in smart homes. The simulator is implemented based on ECHONET Lite, a leading smart home protocol in Japan.

Keywords: smart home simulator, network simulator, network dataset

1. Introduction

As stated in Ref. [1], home network (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.

By utilizing the Internet of Thing (IoT) and ambient assisted living (AAL) concepts, the HN will change to a smart home network where enormous numbers of home appliances can connect and communicate to support the quality of life for people who live in the smart home. The HN is more and more complicated because it changes from home appliances that are operated by humans into being operated by home automation services such as home energy management services and ambient assisted living services. Meanwhile, most of the future users in smart homes are general users without or with insufficient knowledge of network management. Therefore, there is a need to provide intelligent **operations, administration, and maintenance** (OAM) services in order to manage the smart home networks.

The basic network model of smart homes referred from the ITU-T Y.4113 [2] is visualized in **Fig. 1**. Essentially, the *Access Network*, *Core Network*, and *IoT Application Server* are managed by service providers with experienced operators. Contrarily, the *IoT Area Network* is managed by naive smart home users. Recently, machine learning (ML) evolution has achieved breakthroughs in several domains such as computer vision, speech recognition, self-driving cars, and also network management [3]. Unlike the policy-based network management approach (PBNM) [4], where computers can consistently perform repetitive and well-defined policies provided by the network operators, machine learning-based network management (MLBNM) can additionally generate policies by learning from network operator perspectives [5]. ML-based management approaches have achieved promising results in network traffic prediction [6], [7], [8], network fault prediction and detection [9], [10], [11], network security [12], and so on. The critical success of ML-based approaches is the availability of dataset [13], and lack of data is the biggest barrier to build ML-based OAM solutions for smart homes.

A network dataset such as network traffic, logs from network devices could be collected by deploying a real network testbed, a network simulation, or a synthetic environment [14]. Since simu-

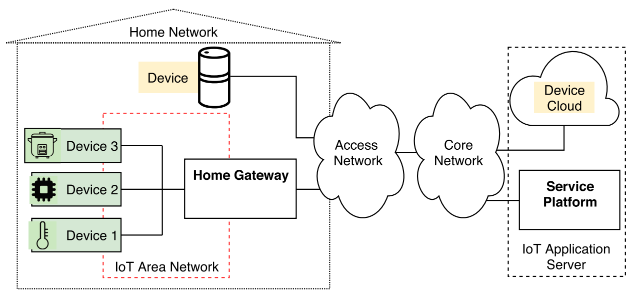


Fig. 1 Basic model of IoT network in the smart home contexts.

1 Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923–1211, Japan
a) cupham@jaist.ac.jp
b) m-yoshi@jaist.ac.jp
c) khoapho@jaist.ac.jp
d) ylim@jaist.ac.jp
e) ytan@jaist.ac.jp

lation achieves high flexibility and low cost in terms of time and money, this research introduces a network simulator to generate datasets of the IoT Area Network. The main contributions of this work include the following:

1 Propose and implement a smart home network simulator that can (i) simulate network traffic from various of smart home services; (ii) simulate network traffic of normal devices and faulty devices;
2 Propose a solution for data processing which extracts features from raw network traffic data.
3 To verify the proposed solution, ML solutions are investigated with the dataset generated from the proposed solutions.

2. Related Work

2.1 Target Home Network Protocol: ECHONET Lite

ECHONET Lite is an open international communication protocol that supports layers 5 to 7 of the OSI model and defines parts of the application and standard command systems for electrical appliances [15]. The ECHONET Lite protocol satisfies all requirements of a protocol of the IoT Area Network, as stated in the ITU-T Y.2070 [16] and ITU-T Y.4113 [2]. In the scope of this paper, the ECHONET Lite is the target protocol.

As shown in **Fig. 2** (a), the *Communication Middleware* [17] is a part of the specification which defines the frame format for communication and basic sequences of an ECHONET Lite node. Basic sequences of an ECHONET Lite node that cover operational aspects of a node, such as a sequence for node start-up, the sequence for node control, and the sequence for receiving a request, are also defined in the specification.

As illustrated in Fig. 2 (b), 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 Object*. A device object represents a logical device which is classified into seven groups and 113 classes of devices in the latest English specification released in 2018 [18]. Device objects offer a standardized method to represent device resources and services via a list of *Property* and constraints for each property.

2.2 Smart Home Simulator

In Refs. [19], [20] and [21], authors 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.

In Ref. [22], a simulated 3D smart home is proposed. It allows simulating operations and communications of devices in smart homes. It simulates the device's communication using the UPnP protocol and supports a flexible mechanism to add more devices by connecting commercial devices into the virtual space. However, abnormal behaviors of devices are not yet considered.

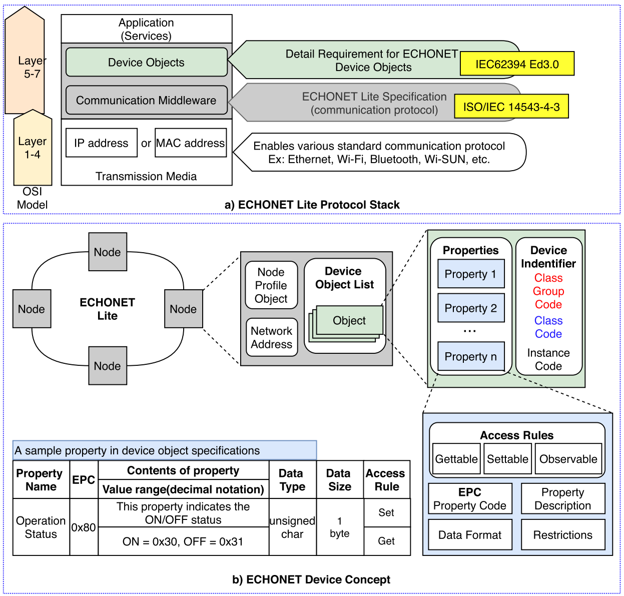


Fig. 2 ECHONET Lite Protocol Stack(a) and Device Object(b).

Table 1 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, parsing, sending and listening for ECHONET Lite packets in Node.js. It allows creating and managing device objects
OpenECHO	Middleware supports creating, parsing, sending and listening for ECHONET Lite packets in Java. Also, It allows creating and managing device objects

2.3 ECHONET Lite Emulator

Since 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 ECHONET Lite software development kit (SDK) referred from the HEMS (ECHONET Lite) Interoperability Test Center [*1] is shown in **Table 1**.

Currently, ECHONET Lite is getting more attention, especially in European countries by a collaboration project [23], these tools and emulators are helpful to get started since ECHONET Lite equipment is not accessible 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.

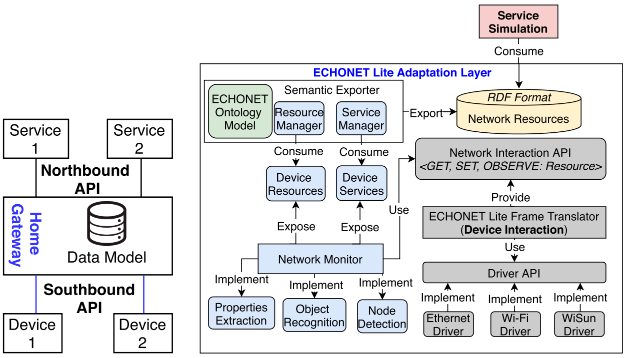
3. Network Simulator

Essentially, the network simulator involves two main components: the home gateway (HGW) and devices connected to the HGW.

3.1 Home Gateway

In the HN, the HGW is a central point that manages and rep-

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



(a) HGW Concept (b) HGW Abstraction Layer

Fig. 3 Home Gateway Concept & ECHONET Lite HGW.

Table 2 Requirements for device.

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 collection request sent from the home gateway. Moreover, it is required to check the status of device and report in the case of failure

resents devices that are connecting to the HGW. Smart home services operate devices via the HGW. Therefore, the network traffic between the HGW and devices reflects these services and can be utilized to diagnose the health of the network.

As visualized in **Fig. 3** (a), the HGW provides the abstraction between (i) the southbound interface that manages the network of connected devices and (ii) the northbound interface that allows service interactions. In Ref. [24], an adaptation layer that provides the network abstraction for the ECHONET Lite protocol has been implemented (Fig. 3 (b)).

The HGW of the simulator is implemented by simulating service scenarios on the top of this adaptation layer.

3.2 Device Emulator

Requirements of a device in the HN, as stated in the ITU-T Y.2070 is summarized in **Table 2**. Thus, the device emulator (DE) is designed while keeping in mind the following points

- Decoupling the *Device Objects* and *Communication Middleware* (in Fig. 2) to improve flexibility when simulating new devices.
- Supporting a mechanism to simulate faulty devices.
- To reduce memory usage, the 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 Ref. [18]).
- Be able to simulate a network with hundreds of devices.

The overview of the ECHONET Lite DE is illustrated in **Fig. 4**.

The purpose of this DE is to take a device configuration file (CF) as input and create an ECHONET Lite node which behaves precisely the same to a commercial device. The proposed DE includes two main components that cover two parts of the ECHONET Lite specifications (Fig. 2 (a)) (i) device object configuration and (ii) middleware and a mechanism to interact with

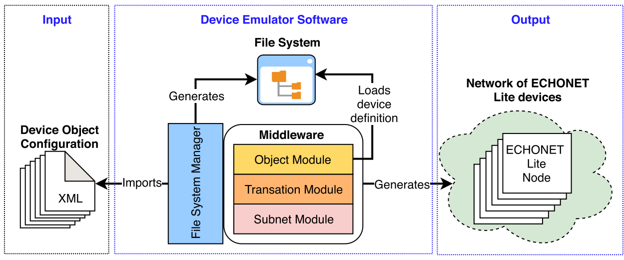


Fig. 4 ECHONET lite device emulator overview.

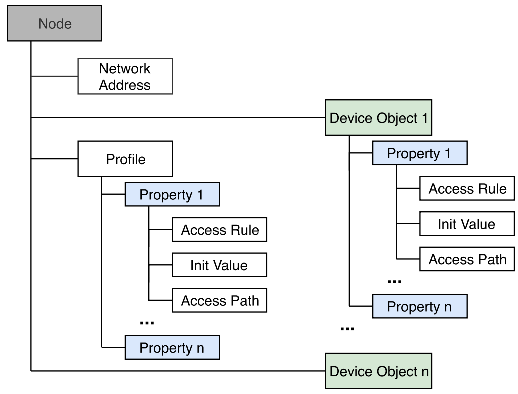


Fig. 5 Device object configuration structure.

the simulated device via a file system (FS).

- Device object configuration (DOC) is a xml [25] document which provides device identification, device resources and services. The structure of the DOC is visualized in **Fig. 5**. 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 (i) device resources are properties with initial values and (ii) device services are specified by *Access Rule*. Each property is attached with a unique *Access Path*, it is a relative path of the property in the FS. The property value can be updated using this path.
- Middleware (MW) represents the standard operating procedure of an ECHONET Lite node including start-up (restart) sequence, request handling sequence, and notify sequence which are required by the ECHONET Lite protocol stack. The proposed MW supports three layers: (i) *Object Module* imports the 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 ECHONET Lite frame into data and vice versa.

A simulated device can be created by defining a DOC and importing to the MW. The FS is respectively generated based on the *Access Path* of each device object. By updating values of properties via the FS, the device status can be simulated by external agents.

3.2.1 Faulty Device Simulation

A faulty device could be simulated by mimicking abnormal behaviors (faults) of commercial devices. In Refs. [26], [27], [28], and [29], faulty behaviors of IoT devices and wireless sensors

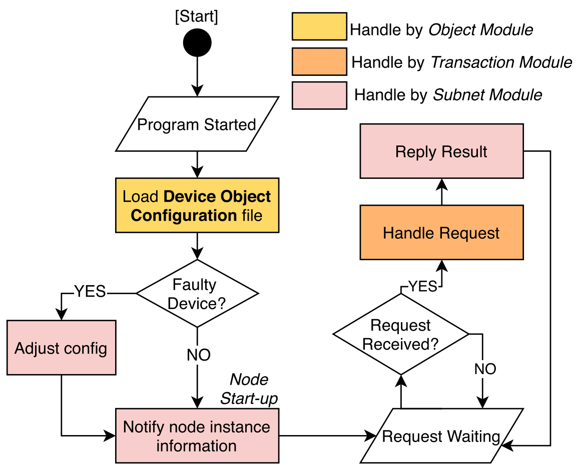


Fig. 6 Simulator middleware (*Humming*) flowchart.

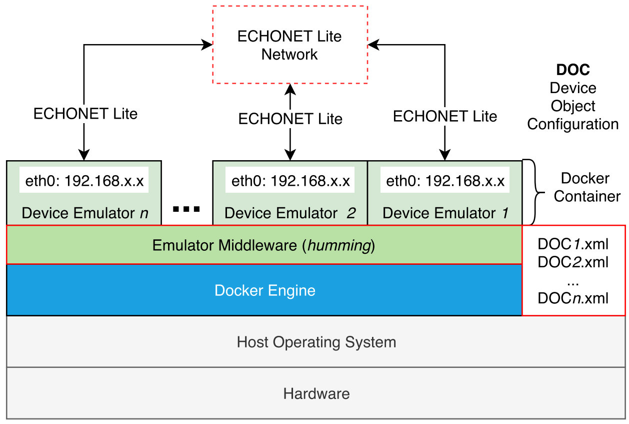


Fig. 7 Deployment overview.

Result: A device simulator as a containerized application
for *DOC file in configuration folder* **do**
  Generate corresponding directory;
  Create a network interface;
  Create a device docker container;
end

Algorithm 1: Device Simulator Deployment

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 Ref. [30], a fault can be categorised into:

- **Crash Fault**: A device completely stops responding. 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
- **Omission Fault**: A device does not reply one or more requests. The *omission fault* shares the same phenomenon as the *crash fault* but since it happens with a possibility of the drop rate is less than 100%.
- **Timing Fault**: Responses of a device occur outside of the specified time interval. The *timing fault* is simulating by adding a time delay before transmitting frames to the *Subnet Module*
- **Response Fault**: A device replies incorrectly either by an incorrect request return value or an incorrect state transition. The *response fault* can be implemented by editing the frame value at the *Subnet Module* to a faulty value before transmitting the outgoing frames.

3.2.2 Implementation

The simulator middleware namely *Humming* is implemented and released as an open source project via github [*2]. *Humming* is a Java implementation that supports all operations of an ECHONET Lite node as stated in Section 2.1. The flowchart of *humming* is shown in **Fig. 6**. 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 Ref. [18] 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) a device keeps rebooting, or (ii) a 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 simulated by sending the *node instance information* after a short arbitrary timing (about one second) to the multicast address of the network. The non-response scenario is simulated 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 (i) replying to a request with a wrong result, (ii) replying to a request by a non-ECHONET Lite frame, or (iii) replying to a request by an invalid value.

3.2.3 Deployment

The overview of the deployment model of the DE is as in **Fig. 7**.

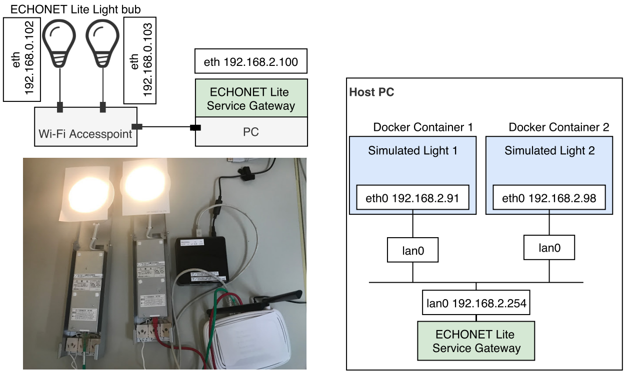
To support an easy and scalable deployment, each DE is deployed as a docker container by the script in Algorithm 1. By utilizing the deployment script, a collection of DOC is mappable to a network of ECHONET Lite nodes.

3.2.4 Evaluation

Firstly, an experiment to evaluate the operational aspects of simulated devices and commercial devices is conducted as in **Fig. 8**.

To verify whether the simulated device can mimic commercial device operations, an *ECHONET Lite HGW* that supports ambient assisted living services [31] is deployed together with

*2 https://github.com/ymakino/humming



a) Deployment of Real ECHONET Lite Devices b) Deployment of Device Simulator

Fig. 8 Experiment configuration.

Table 3 Response time of commercial and simulated devices.

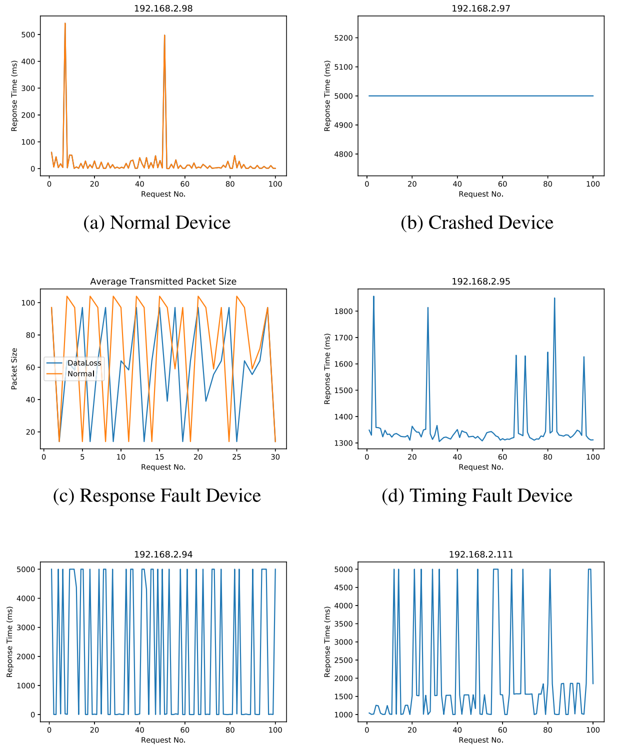
	ECHONET Lite Light 1	ECHONET Lite Light 2	Simulated Light 1	Simulated Light 2
Time to detect device	1067 (ms)	1080 (ms)	1027 (ms)	1048 (ms)
Time to get Profile Object	8 (ms)	7 (ms)	4 (ms)	3 (ms)
Time to get Device Object	20 (ms)	18 (ms)	12 (ms)	13 (ms)
Time to set (ON/OFF)	82 (ms)	76 (ms)	35 (ms)	36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

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 Fig. 8 (a) used as commercial devices. Two simulated lighting devices are configured the same to commercial devices in terms of a number of properties and initial data of each property.

Table 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 **1,000 ms** after multi-casting the request. It requires an additional **13 ms** to detect the second commercial device 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 ms for the *GET* operation and less then 50 ms for the *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, this time variance 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 well. The result shows that emulated devices and commercial devices behaved in the same manner in responding to requests from the HGW.

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



(a) Normal Device (b) Crashed Device (c) Response Fault Device (d) Timing Fault Device (e) Omission Fault Device (Rate 50%) (f) Drop rate 30% and Delay 1000 ms

Fig. 9 Faulty device and normal device response time.

The average amount **100 MB** of memory is required to simulate an ECHONET node.

3.2.5 Faulty Device Evaluation

The response time of 100 requests of emulated devices is visualized in **Fig. 9** and the response timeout **5,000 ms** will be applied in the case of no response. In Fig. 9 (b), the response time is **5,000 ms** for all requests which mimics the crashed fault. In Fig. 9 (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 Fig. 9 (d), the response time equals to **Added Delay Time (1,300 ms) + Normal Response Time**. In Fig. 9 (e), the response timeout is reported with a rate to mimic the omission fault. Several faults can also be combined in one single device as in Fig. 9 (f).

3.3 Network Simulator Deployment Options

Network traffic could be categorized into

- **Machine Generated Traffic** (MGT): the traffic generated only by the device interactions such as periodic data report communication from sensors , or periodic data 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.

The MGT simulation is simply achieved by (i) defining devices in terms of name, number, and configuration, (ii) configuring the

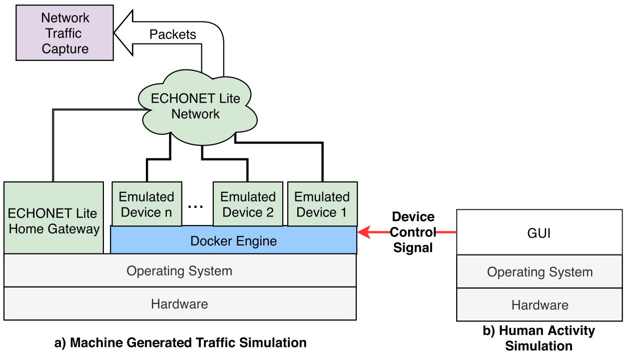


Fig. 10 Home network simulation deployment.

HGW by specifying operation scenarios, (iii) deploying devices, and (iv) setting a traffic monitor to collect the traffic. Thus, the MGT simulation could be done by deploying the DE and the HGW as in **Fig. 10** (a).

Essentially, in the MGT simulation, the traffic is generated by the interaction between the HGW and emulated devices, then 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 could be implemented by extending the *Human Activity Simulation* to operate devices and generate the traffic according to activities as in Fig. 10. As this research aims to provide OAM services, only the MGT is enough to fulfill data generation requirements. However, the proposed simulator also supports HGT by providing APIs for a *human activity simulation* to interact with emulated devices via the file system as in Fig. 10 (b).

4. Network Traffic Dataset Generation

Raw traffic which is in the form of network packets is usable for network forensic investigation, however it is not possible to use the raw traffic as input to build the ML based solutions. Therefore, a method to process and label captured packets will be introduced in this section.

Flow-based approaches for anomaly detection and traffic classification show the potential to achieve low time and memory overhead [32], [33]. Generally, a unidirectional flow (uniFlow) [34] 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. Unlike the unidirectional flow that represents packets flowing in one direction only, a bidirectional (Biflow) [35] represents packets flowing two directions between endpoints on a network. The *Biflow* more accurately describes behaviors and gives more insight information of a network system [35], [36].

A network traffic flow generator, namely *NetFlowMeter* [37], [38], has been widely applied to generate bidirectional flows of network traffic datasets such as a dataset for android malware [39], a dataset for DDoS attack detection [40], a dataset for intrusion detection [41], and so on. However, the situation is different in the IoT Area Network where devices are usually low-power devices with the wake-up mechanism. Moreover, the connection direction of the *NetFlowMeter* is decided by a timely order only, but in the IoT Area Network, it is required to have the direction decided by the initiator (the HGW). Additionally, the *NetFlowMeter* does not support the usage of the multicast address which is essential in the IoT Area Network.

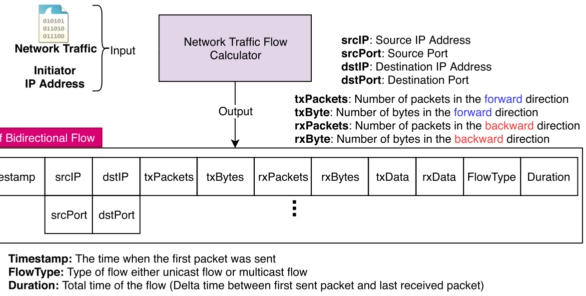


Fig. 11 Network traffic flow calculator.

Table 4 Basic attributes of a *Biflow* 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 overview of the proposed network flow calculator, namely *flowCal*, for the IoT Area Network is illustrated as in **Fig. 11**.

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. 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* as in **Table 4**. The quadruple of *Source IP address, Source Port, Destination IP address, Destination Port* could be utilized to label the flow based on the description of emulated devices. For example, an emulated device with IP address *192.168.2.95* and port *3600* is a *Timing Fault Device* could be used to label bidirectional flows contain this IP address as *Timing Fault* flows.

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 that are derivable from basic attributes such as *Minimum, Mean, and Maximum packet size in forward and backward direction* could be calculated and included in the output.

Table 5 Device configuration of the home network simulator.

Device Object	Total	Normal Devices	Faulty Device				
			Response Fault	Timing Fault	Omission Fault	Timing& Omission Fault	All Fault 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**

5. Application: Network Traffic Classification

To verify the feasibility and usability of the proposed simulator, a ML-based application is introduced in this section. The application is to classify network traffic which is trained using the dataset generated from the proposed simulator.

5.1 Device Emulator Configuration

The summaries of devices of the emulator are in **Table 5**.

In this experiment, a total of **246** device objects are emulated by **138** ECHONET Lite nodes and those devices are reflecting real operating ECHONET Lite devices (device properties and initial data are loaded by values taken from real devices) from a testbed called the *iHouse* [42]. Besides mapping real physical devices into emulated devices, faulty devices are also emulated with the ratio of normal device and faulty device 1:1 (50%:50%).

5.2 Network Simulator Configuration

The last part of the simulator is the simulation of services (that utilize the emulated device) via the configuration of the HGW. In this experiment, the HGW is configured for a context-aware application [43] as follows:

- The HGW sends a *Node Finding Message* to the multicast address of the network during start-up time and re-sends every **2 minutes** to detect newly joined devices and left-the-network devices
- The HGW sends requests to get managed device information (2 requests: a request to get the *Profile Object* and a request to get the *Device Object*) at an interval of every **10 seconds**.

In this experiment, the HN simulator is deployed by utilizing the infrastructures of Hokuriku StarBED [*3] in **22** hours and about **12,674,778** exchanged packets of the MGT are captured by the HN simulator.

5.3 Dataset Generation

The bidirectional flows are exported from the captured raw packets and the network simulator configuration using the proposed *flowCal* as follow:

- Number of *Multicast Flow* = 30 (Flow/Hour) * 138 (Nodes) * 22 (Hours) = **91,080** (flow)
- Number of *Unicast Flow* = 360 (Flow/Hour) * 138 (Nodes) * 22 (Hours) = **1,092,960** (flow)

According to the device emulator configuration, the total number of **1,184,040** flows are labeled into 6 categories: *Normal Flow*, *Response Fault Flow*, *Timing Fault Flow*, *Omission Fault Flow*, *Timing and Omission Fault Flow*, and *All Fault Combined Flow* to form a network dataset.

[*3] http://starbed.nict.go.jp/

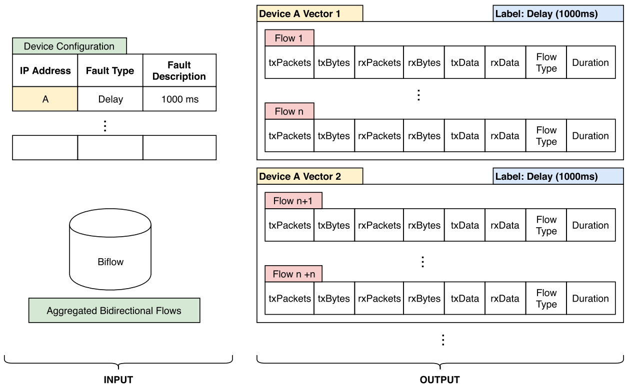


Fig. 12 Data feature extraction overview.

5.4 Machine Learning Based Model Construction

5.4.1 Feature Extraction

The aggregated bidirectional flows from the dataset are reflecting device behaviors. 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 the device behavior even though in the real deployment. In this dataset, the HGW periodically multicasts the *Node Finding Message* at the rate of **2** minutes. Therefore, all extracted **13** flows (1 unicast and 12 multicast) between two multicast requests could be clustered into a vector that reflects the 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 **13** attributes as in Table 4, 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, the remaining **8** attributes are usable as device features and combined as an input vector which represents for the device. Because the input vector represents a device identified by the IP address, it can be labeled using dataset descriptions. The conceptual diagram which **Input** and **Output** of the feature extraction process is illustrated in **Fig. 12**.

In this paper, the Principal Component Analysis (PCA) [44] is applied to reduce high-dimension vectors (13 (flow) * 8 (attributes/flow) = 104 dimensions) into "visualizable" **2-dimension** vectors in order to understand the data distribution.

Since it is impossible to visualize **98,670** samples, a random set of samples (**0.2**%) is selected and visualized as in **Fig. 13**. At first glance, data points of devices that have the *timing fault* are separated from the points of *normal* devices. The distances of the *timing fault* samples respect to its delayed time. The data points

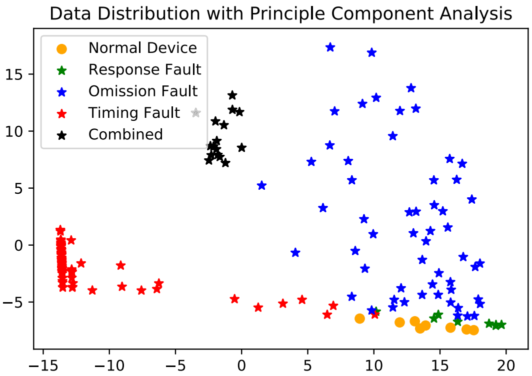


Fig. 13 Data distribution with PCA.

of devices have the *omission fault* are mixed up with *normal* devices.

5.4.2 Experiment

In this section, the performance of ML methods that are built using the dataset is investigated. Each ML method has own purpose to solve specific problems. The selected methods include:

- **Decision Tree** (DT) [45] which is widely used to demonstrate the rule-based approach [46].
- **Support Vector Machine** (SVM) [47] which is the best linear classifier approach [48].
- **Artificial Neural Network** (ANN) [49] which is a general non-linear classifier that achieves huge success in many real-world problems [50].

All experiments are conducted using the computer infrastructure of the Hokuriku starBED. The configuration information of the computer which was used to train and test ML models during the experiment is as follows:

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

5.4.3 Results

The average accuracy of predicting six classes of traffic: *Normal, Response Fault, Timing Fault, Omission Fault, Combined of Timing and Omission Fault, and All Fault Combined* after running ten times with **Z-core** data normalization [51] is:

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

Performances of three investigated methods are summarized in **Table 6**, **Table 7**, and **Table 8**. All three models achieved high accuracy in classifying *normal* devices and devices with *response fault* from the rest of faulty devices. However, the accuracies of detecting *omission fault* and *omission-related fault* device are low. The artificial neural network achieves the best performance with data normalization.

The ANN made the wrong prediction for *omission* fault as it is similar to a fault that combines both the omission fault and the timing fault.

The SVM and DT made the wrong prediction for omission-related faults because they are not linear distributions.

Table 6 Artificial neural network normalized confusion matrix.

GroundTruth	Normal	Reponse Fault	Timing fault	Omission Fault	Timing & Omission Fault	All Fault Combined
Normal	**99.87**	0.02	0	0.08	0	0.03
Reponse Fault	0.28	**99.72**	0	0	0	0
Timing Fault	0.22	0	**95.09**	0	3.67	1.02
Omission Fault	1.27	0.33	0.19	**90.51**	6.2	1.5
Timing & Omission Fault	0.29	0.04	1.29	1.42	**92.36**	4.59
All fault Combined	0	0.08	0.08	4.67	2.03	**93.15**

Table 7 Decision tree normalized confusion matrix

Groundtruth	Normal	Reponse Fault	Timing Fault	Omission Fault	Timing & Omission Fault	All Fault Combined
Normal	**98.34**	0.12	0.02	1.52	0	0
Reponse Fault	0.47	**98.97**	0	0.56	0	0
Timing fault	0.71	0	**89.69**	3.14	6.46	0
Omission Fault	11.42	0.38	0.28	**85.29**	1.6	1.03
Timing & Omission Fault	0	0	2.34	3.17	**87.43**	7.06
All Fault Combined	0	0	0.23	5.03	10.82	**83.92**

Table 8 Support vector machine normalized confusion matrix.

Groundtruth	Normal	Reponse Fault	Timing fault	Omission Fault	Timing & Omission Fault	All Fault Combined
Normal	**97.04**	0.54	0.31	2.06	0.05	0
Reponse Fault	3	**96.9**	0	0.1	0	0
Timing fault	0	0	**91.8**	0.55	6.6	1.05
Omission Fault	11.46	3.48	6.43	**71.58**	3.57	3.48
Timing & Omission Fault	1.21	1.96	6.47	4.38	**80.67**	5.3
All Fault Combined	0.3	0	1.65	12.01	4.88	**81.16**

5.4.4 Discussion

All investigated ML methods achieves (i) good performances in classifying *Normal* and *Response Fault*, (ii) bad performances in classifying *Omission Fault* and omission related faults (Timing & Omission Fault, All Fault Combined), and (iii) average performances in classifying *Timing Fault*. Results show the high performance of the linear distribution of samples and the low accuracy of the non-linear distribution of samples.

Since the more flows collected, the better judgment of device behaviors, we can cluster flows of a device for a day or a week for the training dataset. However, in the 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 the 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.

In this experiment, a sample that represents a device is a 104-dimension vector. The sample is calculated every 2 minutes where (i) one multicast flow and 12 unicast flows are combined and (ii) eight features of flow are extracted from a flow. By combining the traffic of every 2 minutes, it achieves a shorter time to classify the targeted traffic in a real deployment. However, the information collected during the 2-minute interval may not be enough to represent a device. Therefore, there is a trade-off between the high accuracy and the high overhead.

6. Conclusion

This research proposed a smart home network simulator to generate the IoT Area Network traffic so as to pave the way for integrating AI-based solutions for home network management. Since the ECHONET Lite protocol fulfills all the requirements of the IoT Area Network, the ECHONET Lite is the target protocol of the simulator. The proposed simulator is able to simulate various types of services by extending the southbound interfaces on the top of an ECHONET Lite abstraction layer. Moreover, the simulator supports mechanism to simulate commercial ECHONET Lite devices and faulty devices as well. Four types of faults: 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 **100 MB** and CPU usage (**0.15%**) for a node are suitable to deploy on a large scale.

The evaluation of the network traffic generated by deploying the proposed simulator has been done. The network traffic in the dataset (in the form of captured raw packets) is aggregated into bidirectional flows that reflect the device-gateway interactions by the proposed network flow calculator, namely *Flowcal*. The *Flowcal* is customizing for the IoT Area Network that supports the appointment of flow direction initiator and multicast flows. Three ML methods: 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%) in predicting device faults based on network traffic. Results proved the feasibility and usability of the dataset generated from the proposed simulator in building ML based solutions.

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 the high reliability, availability, and scalability platform that supports a big data analytics framework such as PNDA [*4] is desired as future work. The mechanism to retrain the model with the online data collection is also extensible as future work.

[*4] http://pnda.io/

References

[1] International Telecommunication Union: *Architecture of MediaHomeNet* (2007).

[2] International Telecommunication Union: *Requirements of the network for the Internet of things* (2016).

[3] De Donato, W., Pescapé, A. and Dainotti, A.: Traffic identification engine: An open platform for traffic classification, *IEEE Network*, Vol.28, No.2, pp.56–64 (online), DOI: 10.1109/MNET.2014.6786614 (2014).

[4] Ding, J.: *Advances in Network Management*, Auerbach Publications (2016).

[5] Boutaba, R., Salahuddin, M.A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F. and Caicedo, O.M.: A comprehensive survey on machine learning for networking: evolution, applications and research opportunities, *Journal of Internet Services and Applications*, Vol.9, No.1, p.16 (online), DOI: 10.1186/s13174-018-0087-2 (2018).

[6] Li, Y., Liu, H., Yang, W., Hu, D. and Xu, W.: Inter-data-center network traffic prediction with elephant flows, *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pp.206–213 (online), DOI: 10.1109/NOMS.2016.7502814 (2016).

[7] Zhitang Chen, Jiayao Wen and Yanhui Geng: Predicting future traffic using Hidden Markov Models, *2016 IEEE 24th International Conference on Network Protocols* (*ICNP*), pp.1–6 (online), DOI: 10.1109/ICNP.2016.7785328 (2016).

[8] Poupart, P., Chen, Z., Jaini, P., Fung, F., Susanto, H., Geng, Y., Chen, L., Chen, K. and Jin, H.: Online flow size prediction for improved network routing, *2016 IEEE 24th International Conference on Network Protocols* (*ICNP*), pp.1–6 (online), DOI: 10.1109/ICNP.2016.7785324 (2016).

[9] Wang, Z., Zhang, M., Wang, D., Song, C., Liu, M., Li, J., Lou, L. and Liu, Z.: Failure prediction using machine learning and time series in optical network, *Opt. Express*, Vol.25, No.16, pp.18553–18565 (online), DOI: 10.1364/OE.25.018553 (2017).

[10] Rao, S.: Operational Fault Detection in cellular wireless base-stations, *IEEE Trans. Network and Service Management*, Vol.3, No.2, pp.1–11 (online), DOI: 10.1109/TNSM.2006.4798311 (2006).

[11] Qader, K., Adda, M. and Al-Kasassbeh, M.: Comparative analysis of clustering techniques in network traffic faults classification, *International Journal of Innovative Research in Computer and Communication Engineering*, Vol.5, No.4, pp.6551–6563, DOI: 10.15680/IJIRCCE.2017.0504001 (2017).

[12] Buczak, A.L. and Guven, E.: A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection, *IEEE Communications Surveys Tutorials*, Vol.18, No.2, pp.1153–1176 (online), DOI: 10.1109/COMST.2015.2494502 (2016).

[13] Roh, Y., Heo, G. and Whang, S.E.: A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective, *IEEE Trans. Knowledge and Data Engineering*, p.1 (online), DOI: 10.1109/TKDE.2019.2946162 (2019).

[14] Grochla, K. and Naruszewicz, L.: Testing and Scalability Analysis of Network Management Systems Using Device Emulation, *Computer Networks*, Kwiecie'n, A., Gaj, P. and Stera, P. (Eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp.91–100 (2012).

[15] Umejima, M.: Japan's Power Meter Deployment with ECHONET Lite over IPv6, *New Breeze - Quarterly of the ITU Association of Japan*, Vol.27, No.2, pp.12–13 (2015).

[16] International Telecommunication Union: *Semantics based requirements and framework of the Internet of things* (2016).

[17] ECHONET CONSORTIUM: *Communication Middleware Specifications* (2018).

[18] ECHONET CONSORTIUM: *Detailed Requirements for ECHONET Device Objects* (2018).

[19] Helal, A., Cho, K., Lee, W., Sung, Y., Lee, J.W. and Kim, E.: 3D Modeling and Simulation of Human Activities in Smart Spaces, *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*, pp.112–119 (online), DOI: 10.1109/UIC-ATC.2012.35 (2012).

[20] Synnott, J., Chen, L., Nugent, C.D. and Moore, G.: The creation of simulated activity datasets using a graphical intelligent environment simulation tool, *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp.4143–4146 (online), DOI: 10.1109/EMBC.2014.6944536 (2014).

[21] Alshammari, N., Alshammari, T., Sedky, M., Champion, J. and Bauer, C.: OpenSHS: Open Smart Home Simulator, *Sensors*, Vol.17, No.5 (online), DOI: 10.3390/s17051003 (2017).

[22] Nishikawa, H., Yamamoto, S., Tamai, M., Nishigaki, K., Kitani, T., Shibata, N., Yasumoto, K. and Ito, M.: UbiREAL: Realistic Smartspace Simulator for Systematic Testing, *UbiComp* (2006).

[23] Bruno, B., Chong, N.Y., Kamide, H., Kanoria, S., Lee, J., Lim, Y., Pandey, A.K., Papadopoulos, C., Papadopoulos, I., Pecora, F., Saffiotti, A. and Sgorbissa, A.: The CARESSES EU-Japan Project: Making Assistive Robots Culturally Competent, *Ambient Assisted Living - Italian Forum 2017, 8th Italian on Ambient Assisted Living Forum, ForItAAL 2017*, pp.151–169 (online), DOI: 10.1007/978-3-030-04672-9_10 (2017).

[24] Pham, V.C., Lim, Y., Sgorbissa, A. and Tan, Y.: An Ontology-driven ECHONET Lite Adaptation Layer for Smart Homes, *Journal of Information Processing*, Vol.27, pp.360–368 (online), DOI: 10.2197/ipsjjip.27.360 (2019).

[25] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E. and Yergeau, F.: Extensible markup language (XML) 1.0 (2000).

[26] Muhammed, T. and Shaikh, R.: An Analysis of Fault Detection Strategies in Wireless Sensor Networks, *Journal of Network and Computer Applications*, Vol.78, pp.267–287 (online), DOI: 10.1016/j.jnca.2016.10.019 (2017).

[27] Noshad, Z., Javaid, N., Saba, T., Wadud, Z., Saleem, M., Alzahrani, M. and Sheta, O.: Fault Detection in Wireless Sensor Networks through the Random Forest Classifier, *Sensors*, Vol.19, p.1568 (online), DOI: 10.3390/s19071568 (2019).

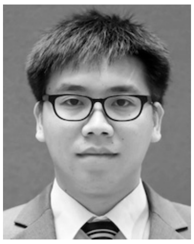
[28] Sharma, A., Golubchik, L. and Govindan, R.: Sensor Faults Detection Methods and Prevalence in Real-World Datasets, *TOSN*, Vol.6 (2010).

[29] Chakraborty, T., Nambi, A.U., Chandra, R., Sharma, R., Swaminathan, M., Kapetanovic, Z. and Appavoo, J.: Fall-curve: A Novel Primitive for IoT Fault Detection and Isolation, *Proc. 16th ACM Conference on Embedded Networked Sensor Systems*, *SenSys '18*, pp.95–107, ACM (online), DOI: 10.1145/3274783.3274853 (2018).

[30] Cristian, F.: Understanding Fault-tolerant Distributed Systems, *Comm. ACM*, Vol.34, No.2, pp.56–78 (online), DOI: 10.1145/102792.102801 (1991).

[31] Pham, V.C., Lim, Y., Tan, Y. and Chong, N.Y.: Support for ECHONET-based smart home environments in the universAAL ecosystem, *2018 IEEE International Conference on Consumer Electronics* (*ICCE*), pp.1–4 (online), DOI: 10.1109/ICCE.2018.8326218

(2018).
[32] Noirie, L., Dotaro, E., Carofiglio, G., Dupas, A., Pecci, P., Popa, D. and Post, G.: Semantic networking: Flow-based, traffic-aware, and self-managed networking, *Bell Labs Technical Journal*, Vol.14, No.2, pp.23–38 (online), DOI: 10.1002/bltj.20371 (2009).
[33] Park, J., Tyan, H. and Kuo, C.J.: Internet Traffic Classification for Scalable QOS Provision, *2006 IEEE International Conference on Multimedia and Expo*, pp.1221–1224 (online), DOI: 10.1109/ICME.2006.262757 (2006).
[34] Claise, B.: Cisco Systems NetFlow Services Export Version 9, RFC 3954, RFC Editor (2004), available from ⟨http://www.rfc-editor.org/rfc/rfc3954.txt⟩.
[35] Trammell, B. and Boschi, E.: Bidirectional Flow Export Using IP Flow Information Export (IPFIX), RFC 5103, RFC Editor (2008), available from ⟨http://www.rfc-editor.org/rfc/rfc5103.txt⟩.
[36] Minarik, P., Vykopal, J. and Krmicek, V.: Improving Host Profiling with Bidirectional Flows, *2009 International Conference on Computational Science and Engineering*, Vol.3, pp.231–237 (online), DOI: 10.1109/CSE.2009.23 (2009).
[37] Lashkari, A.H., Draper-Gil, G., Mamun, M.S.I. and Ghorbani, A.A.: Characterization of Tor Traffic using Time based Features, *ICISSP*, pp.253–262 (2017).
[38] Draper-Gil, G., Lashkari, A.H., Mamun, M.S.I. and Ghorbani, A.A.: Characterization of encrypted and vpn traffic using time-related, *Proc. 2nd International Conference on Information Systems Security and Privacy* (*ICISSP*), pp.407–414 (2016).
[39] Taheri, L., Kadir, A.F.A. and Lashkari, A.H.: Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls, *2019 International Carnahan Conference on Security Technology* (*ICCST*), pp.1–8, IEEE (2019).
[40] Sharafaldin, I., Lashkari, A.H., Hakak, S. and Ghorbani, A.A.: Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy, *2019 International Carnahan Conference on Security Technology* (*ICCST*), pp.1–8, IEEE (2019).
[41] Sharafaldin, I., Lashkari, A.H. and Ghorbani, A.A.: Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization, *ICISSP*, pp.108–116 (2018).
[42] Lim, Y. and Tan, Y.: Time Delay Modeling for Energy Efficient Thermal Comfort Control System in Smart Home Environment, *Computational Science and Technology*, Alfred, R., Iida, H., Ag Ibrahim, A.A. and Lim, Y. (Eds.), Singapore, Springer Singapore, pp.42–52 (2018).
[43] Pham, V.C., Lim, Y., Bui, H., Tan, Y., Chong, N.Y. and Sgorbissa, A.: An Experimental Study on Culturally Competent Robot for Smart Home Environment, *Advanced Information Networking and Applications - Proc. 34th International Conference on Advanced Information Networking and Applications, AINA-2020*, Barolli, L., Amato, F., Moscato, F., Enokido, T. and Takizawa, M. (Eds.), Advances in Intelligent Systems and Computing, Vol.1151, pp.369–380, Springer (online), DOI: 10.1007/978-3-030-44041-1_34 (2020).
[44] Abdi, H. and Williams, L.J.: Principal Component Analysis, *WIREs Comput. Stat.*, Vol.2, No.4, pp.433–459 (online), DOI: 10.1002/wics.101 (2010).
[45] Quinlan, J.R.: Induction of Decision Trees, *Mach. Learn.*, Vol.1, No.1, pp.81–106 (online), DOI: 10.1023/A:1022643204877 (1986).
[46] Luna, J.M., Gennatas, E.D., Ungar, L.H., Eaton, E., Diffenderfer, E.S., Jensen, S.T., Simone, C.B., Friedman, J.H., Solberg, T.D. and Valdes, G.: Building more accurate decision trees with the additive tree, *Proc. National Academy of Sciences*, Vol.116, No.40, pp.19887–19893 (online), DOI: 10.1073/pnas.1816748116 (2019).
[47] Hearst, M.A.: Support Vector Machines, *IEEE Intelligent Systems*, Vol.13, No.4, pp.18–28 (online), DOI: 10.1109/5254.708428 (1998).
[48] Fernández-Delgado, M., Cernadas, E., Barro, S. and Amorim, D.: Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?, *Journal of Machine Learning Research*, Vol.15, pp.3133–3181 (online) (2014), available from ⟨http://jmlr.org/papers/v15/delgado14a.html⟩.
[49] Priddy, K.L. and Keller, P.E.: *Artificial Neural Networks: An Introduction* (*SPIE Tutorial Texts in Optical Engineering, Vol.TT68*), SPIE-International Society for Optical Engineering (2005).
[50] Abiodun, O.I., Jantan, A., Omolara, A.E., Dada, K.V., Mohamed, N.A. and Arshad, H.: State-of-the-art in artificial neural network applications: A survey, *Heliyon*, Vol.4, No.11, p.e00938 (2018).
[51] Slapnik, M., Istenič, D., Pintar, M. and Udovč, A.: Extending life cycle assessment normalization factors and use of machine learning – A Slovenian case study, *Ecological Indicators*, Vol.50, pp.161–172 (online), DOI: 10.1016/j.ecolind.2014.10.028 (2015).



Van Cu Pham received his B.Eng. degree in software engineering from FPT University in 2014. He received his M.S. and Ph.D. degree in Information Science from the Japan Advanced Institute of Science and Technology (JAIST) in 2016 and 2020, respectively. Currently, he is a postdoc researcher at the Center for Trustworthy IoT Infrastructure, JAIST. His research interests include home network system, home network management, network simulation. He is a member of IEICE and IPSJ.

Yoshiki Makino received his M.S. degree in Information Science from the Japan Advanced Institute of Science and Technology (JAIST). Currently, he is a researcher at JAIST. His research interests include network simulator, smart home and smart community simulator.

Khoa Pho received his B.S. degree in Computer Science from VNU-HCMC University of Science in 2016. He received his M.S. degree in Information Science from the Japan Advanced Institute of Science and Technology (JAIST) in 2018. Currently, he is a Ph.D. student at JAIST. His research interests include Machine Learning and Computer Vision. He is a member of IEEE.

Yuto Lim received his B.Eng. (Hons) and M.Inf. Technology degrees from Universiti Malaysia Sarawak (UNIMAS), Malaysia in 1998 and 2000, respectively. He received his Ph.D. degree in communications and computer engineering from Kyoto University in 2005. He was a visiting researcher at Fudan University in China for two months. During 2005–2009, he was an expert researcher at National Institute of Information and Communications Technology (NICT), Japan. Since 2009, he has been working at Japan Advanced Institute of Science and Technology (JAIST) as an associate professor. His research interests include multihop wireless networks, wireless sensor networks, home networks, wireless mesh networks, heterogeneous wireless networks, network coding, cyber-physical system. He is a member of IEEE, IEICE and IPSJ.

Yasuo Tan received his Ph.D. from Tokyo Institute of Technology in 1993. He joined Japan Advanced Institute of Science and Technology (JAIST) as an assistant professor of the School of Information Science in 1993. He has been a professor since 1997. He is interested in Ubiquitous Computing System especially Home Networking Systems. He is a leader of Residential ICT SWG of New Generation Network Forum, a chairman of Green Grid Platform at Home Alliance, an advisory fellow of ECHONET Consortium, and a member of IEEE, ACM, IPSJ, IEICE, IEEJ, JSSST and JNNS.