A Study of CoT Architecture for Interworking between IoT Networks and Cloud Using IETF CoAP and OCF IoTivity

Songai Xuan

2018

A Thesis

For the Degree of Master of Engineering

A Study of CoT Architecture for Interworking between IoT Networks and Cloud Using IETF CoAP and OCF IoTivity

Songai Xuan

Department of Computer Engineering

Graduate School

Jeju National University

June 2018

Acknowledgment

First and foremost, I would like to show my deepest gratitude to my supervisor, Prof. Do-Hyeun Kim, a respectable, responsible and resourceful scholar, who has provided me with valuable guidance in every stage of the writing of this thesis. Without his enlightening instruction, impressive kindness and patience, I could not have completed my thesis. His keen and vigorous academic observation enlightens me not only in this thesis but also in my future study.

I shall extend my thanks to all the teachers in the department for their kindness and help, they provide me with great convenience in the writing of my thesis. Then I would also like to express my gratitude to my lab mates, I have drawn great inspiration, which is essential to the completion of my thesis. Their spirits always guide me in the whole process of writing.

Last but not least, I would like to express my thanks to my family and my friends for their valuable encouragement and spiritual support during my study.

June, 2018

Songai Xuan

Table of Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
AWS	Amazon Web Services
CoAP	Constrained Application Protocol
CoT	Cloud of Things
CRUDN	Create, Read, Update, Delete and Notify
GAE	Google App Engine
GPRS	General Packet Radio Service
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
M2M	Machine to Machine
MQTT	MQ Telemetry Transport
OBSB	On Board Smart Box
OCF	Open Connectivity Foundation
REST	Representational State Transfer
URIs	Universal Resource Identifiers

Abstract

In the modern intelligent life, data is everywhere, it is necessary to collect the data and extract useful part for analysis. To collect and extract useful data, the Internet of Things (IoT) technology is important, it is able to emerge paradigm that manages billions of devices, gateways, sensors, and actuators connected to the Internet, and support intelligent services based on huge context data. The IoT technology has obtained great development over the last few years and is increasingly influencing various industrial development, the IoT infrastructures and systems have been deployed to various important area, frequently used for building smart environment, such as smart cities and smart homes. Clouds are also wildly used for huge data repository and Internet services in various fields. Besides, building the connection between IoT devices and cloud is very useful and convenient.

In this paper, we present the CoT architecture based on IoT and Cloud using IETF CoAP and OCF IoTivity. And we design and implement five CoT architectures based on the combine of IoT and cloud for connectivity between IoT Networks and cloud. There are five different architectures which are able to collect massive sensing data from IoT devices and then upload the data to cloud for the further analysis.

This paper present first CoT architecture using IETF CoAP for interworking local Cloud and IoT platforms. This architecture connects IoT devices and IoT platforms, and interconnets IoT platform and Local Cloud for collecting huge sensing data. We design and implement the first CoT architecture based on agent for interworking local Cloud and IoT Networks. We use Hadoop Distributed File System (HDFS) for local Cloud. IoT platform is able to upload sensing data to HDFS using IETF CoAP.

Next, we introduce the second CoT architecture based on agent for interworking local Cloud and IoT Networks. This architecture connects IoT devices and local Cloud based on agent directly. And we design and implement the second CoT architecture based on agent for interworking local Cloud and IoT Networks.

We use HDFS for local Cloud. There is an agent that is able to get sensing data from IoT devices and upload the data to HDFS using IETF CoAP.

And, we describe the third CoT architecture using MQTT for interworking global Clouds and IoT networks. And we design and implement the second CoT architecture using AWS, Azure, and Google Cloud used as global cloud. The CoT architecture communicates between global Clouds and IoT networks using MQTT protocol and store the sensing data. And we compare IoT services based on global Clouds for huge context acquisition in large scale IoT networks. The comparison helps users to choose easily IoT service based on Cloud. Hence, it is necessary to collect the context data easily and extract useful part for information analysis and usage in Cloud based on IoT.

Forth, we propose the CoT architecture for global cloud and IoT networks based on proxy. The proxy is able to get sensing data from IoT networks and send to store in cloud (AWS IoT, Azure IoT Hub, and Google Cloud IoT Core) using OCF IoTivity and MQTT protocol. After some experiments we also compare the IoT cloud services of AWS, Azure, and Google Cloud Platform.

Finally, we present the fifth CoT architecture based on IETF CoAP for vehicle monitoring service. And the administer controls and monitors own vehicles, and uploads vehicle state to cloud. Other clients make able to monitor vehicles states and get vehicle state from cloud.

1. Introduction

The Internet-of-Things (IoT), or more prosaically Machine-to-Machine (M2M), has received significant attention lately from both industry and academia as an emerging paradigm that manages billions of devices, gateways, sensors, and actuators connected to the Internet [1]. There is a paper [2] present an overall view of interworking architectures, which enables exposure of various underlying network services for M2M applications running on top of the service layer, such as device triggering, device location, device management, etc. And there is another paper [3] present an introduction of standardized interworking interfaces and procedures based on oneM2M global standards, and tests them through use cases involving multiple IoT service platforms. The interworking involves smart city applications/services running on multiple IoT service layer platforms interoperating with each other. The Internet of Things (IoT) technology has obtained great development over the last few years and is increasingly influencing various industrial development [4]. Recently, encouraged by the likes of Ericsson and Cisco with estimates of 50 billion Internet connected devices by 2020 [5], it is really necessary to learn the interworking between IoT Cloud and IoT Devices. There is a paper [6] present a data storage framework not only enabling efficient storing of massive IoT data, but also integrating both structured and unstructured data. And there is another paper [7] present recent developments in commercial IoT frameworks and furthermore, identify trends in the current design of frameworks for the Internet of Things; enabling massively connected cyber physical systems.

Internet of Things (IoT) infrastructures and systems have been deployed to various important area, frequently used for building smart environment, such as smart cities and smart homes [8]. Smart home is able to automatically sense the changes of home situations, dynamically response corresponding reactions and autonomously help its residents to make more comfortable lives [9]. For a smart home, there could be an IoT-based monitoring system using a tri-level context making model for context-aware services [10], there could be a device-level protection augmented with network-level security solutions to monitor

network activity and detect suspicious behavior [11]. The smart environment makes people's lives faster and more convenient.

IoT services has the constraint of IoT devices in a large-scale network. Be-cause IoT devices has the limited computing resources, memory capacity, energy, and communication bandwidth. Many of these issues could be resolved by employing the Cloud-assisted Internet of Things as it offers large-scaled and on-demand networked computing resources to manage, store, process and share huge IoT data. It is an issue that how to deal with the large amount of information generated by the intelligent environment. Cloud computing is a good choice. Many researchers have already presented some survey of cloud compo-ting, analyzed the key concepts and architecture [12] or introduced the cloud ser-vices of the IT companies [13]. Some researchers analyzed the authenticator-based data integrity verification techniques on cloud and IoT data [14]. The paper [15] presents an approach to the development of Smart Home applications by in-targeting Internet of Things (IoT) with Web services and Cloud computing, their approach focuses on Arduino platform, Zigbee technology, JSON data format, and cloud services.

The work presented in [16] presents a novel multilayered vehicular data cloud platform including an intelligent parking cloud service and a vehicular data mining cloud service by using cloud computing and IoT technologies. Some researchers developed a systematic comparator of the performance and cost of cloud providers called "CloudCmp" [17], which is able to help customers pick a cloud that fits their needs.

In addition to the smart cities and smart homes, it is also useful and meaningful to develop smart cars, there are already some developers combined IoT and vehicles. Some developers developed an IoT system to allow the monitoring and control of parameters of the users' vehicles [18], Or a system which is able to provide a low-cost means of monitoring a vehicle's performance and tracking by communicating the obtained data to a mobile device via Bluetooth [19]. Some other developers focused on providing automatic and efficient electric vehicles charging management system by exploiting the benefits of IoT technology in offering the ubiquitous perception abilities and a real-time interactive view of the physical world by various sensors and radio devices [20]. Some developers paid more attention to detail and proposed a flexible

infrastructure for dynamic power control of electric vehicle battery chargers, the infrastructure dynamically adjusts the electric vehicle battery charger current, according to the power demand of the home wherein the vehicle is plugged [21]. To remote monitoring vehicles, it is able to use a communication service system for vehicle remote monitoring based on the Netty pattern, which is improved on the basis of the traditional Reactor model, the SEDA mode to handle the event, and from protocol analysis model, shared data synchronization and thread pool to optimize the design [22]. It is also to use a distributed system for remote monitoring of vehicle diagnostics and geographical position, which is achieved by using on-board microcomputer system, called on-board smart box (OBSB), general packet radio service (GPRS) and a remote server [23]. Or use a portable road side vehicle monitoring system for vehicle classification, and speed measurement [24]. Or use a remote monitoring system for lithium battery of electric vehicle to improve the real time monitoring ability and safe operation of the electric vehicle lithium battery, save the cost of battery [25].

Recently, more and more developers begin to combine IoT and cloud, some researchers referred it as Cloud of Things (CoT) and propose some key issues along with their respective potential solutions [26]. Some other researchers focused on the implementation of the underlying infrastructure at the basis of the CoT. An ad-hoc architecture and some preliminary background of this challenging view are provided and discussed, identifying guidelines and future directions [27].

CoT is able to be used for the build of smart cities, solve the issue that different IoT ecosystems are not able to communicate between them by browse the semantic annotation of the sensors in the cloud, and innovative services can be implemented and considered by bridging Clouds and IoT [28], or like ClouT project, which is able to make citizens aware of city resources and helping them to use and care them by mean of smart IoT services in the Cloud [29],

For CoT communication, it is able to use smart gateway [30], some gateways enable a lightweight and dense deployment of services, they are able to manage semantic-like things and at the same time to act as an end-point for the presentation of data to users [31]. And for CoT security, is able to use secure trusted

things as a service to reduce majority of the challenges in CoT environment, the main focus is on encryption mechanism with less overhead besides a trust model to enable real time decision making authentic [32].

Figure 1 shows the conceptual model of the whole design for the connectivity between IoT Networks and Cloud presented in this paper. There are four different CoT architectures. The first CoT architecture is for Hadoop and IoT platforms, IoT platform is able to upload sensing data files to Hadoop Distributed File System (HDFS). Hadoop is able to connect many sensor platforms. And each IoT platform is able to connect many Sensor Middlewares, each Sensor Middleware is able to connect many sensors. IoT platform provide sensor information and sensing data storage service. Sensor Middlewares get sensing data from sensors and save the data into database via the service provided by Sensor Platform. Then the agents will request sensing data and sensor information from IoT platform and upload the data to Hadoop.

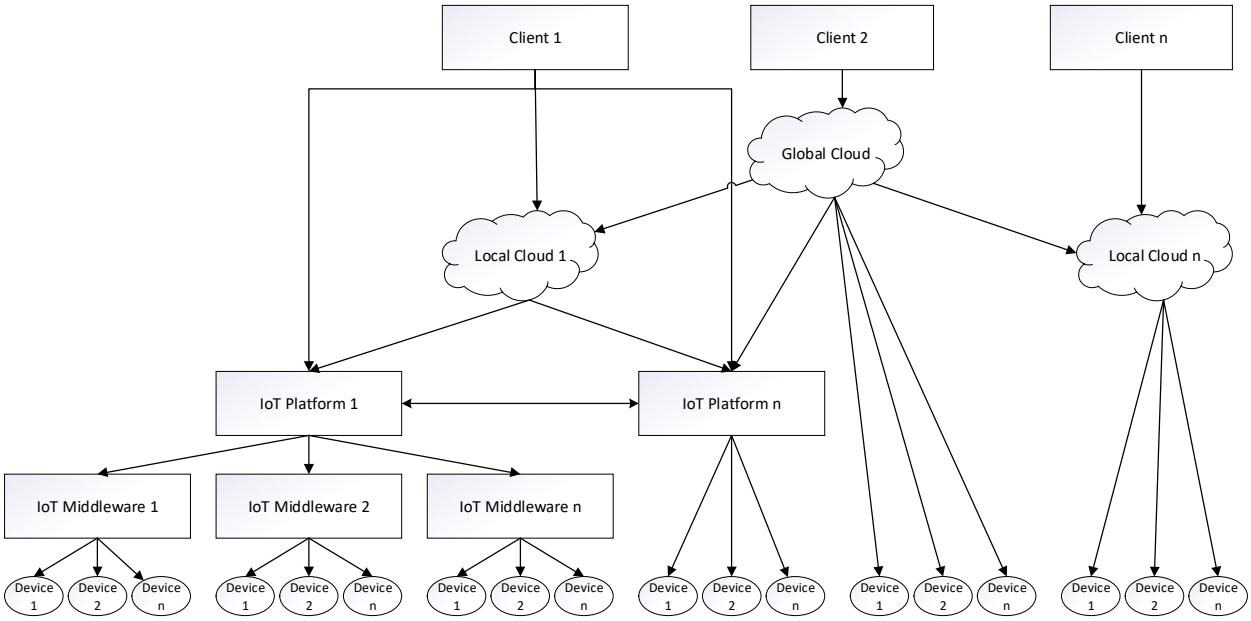


Figure 1. Conceptual model of CoT architecture based on Proxy using IETF CoAP and OCF IoTivity

The second CoT architecture is for Hadoop and IoT Networks, there is an agent that is able to get sensing data from IoT devices and upload the data to Hadoop Distributed File System (HDFS) using CoAP protocol. Hadoop is able to connect many agents; each agent is able to connect many IoT devices and each IoT device

is able to connect many sensors. IoT Device will get sensing data from sensors and send to the agent, and agents will upload sensing data to Hadoop.

The third CoT architecture is for global cloud and IoT networks, AWS, Azure, and Google Cloud is used as global cloud and is able to communicate with IoT networks using MQTT protocol and store the sensing data. The IoT devices are exactly the same, each of them is able to publish message to the topic of IoT cloud (AWS IoT, Azure IoT Hub, and Google Cloud IoT Core) and store the sensing data to cloud. We have also compared and analyzed the performance of three IoT cloud services based on the process and results of the experiment.

The fourth CoT architecture is for global cloud and IoT networks based on proxy using OCF IoTivity and MQTT protocol, the proxy is able to get sensing data from IoT networks and then publish the data as messages to the topic in IoT cloud (AWS IoT, Azure IoT Hub, and Google Cloud IoT Core). After some experiments we also compared the IoT cloud service of AWS, Azure, and Google Cloud Platform.

The fifth CoT architecture is for vehicle monitoring and control service, the client is able to monitor vehicles, control vehicles, and get vehicle state from cloud.

2. Related Work

2.1 Cloud (Hadoop, Azure, AWS, Google Cloud)

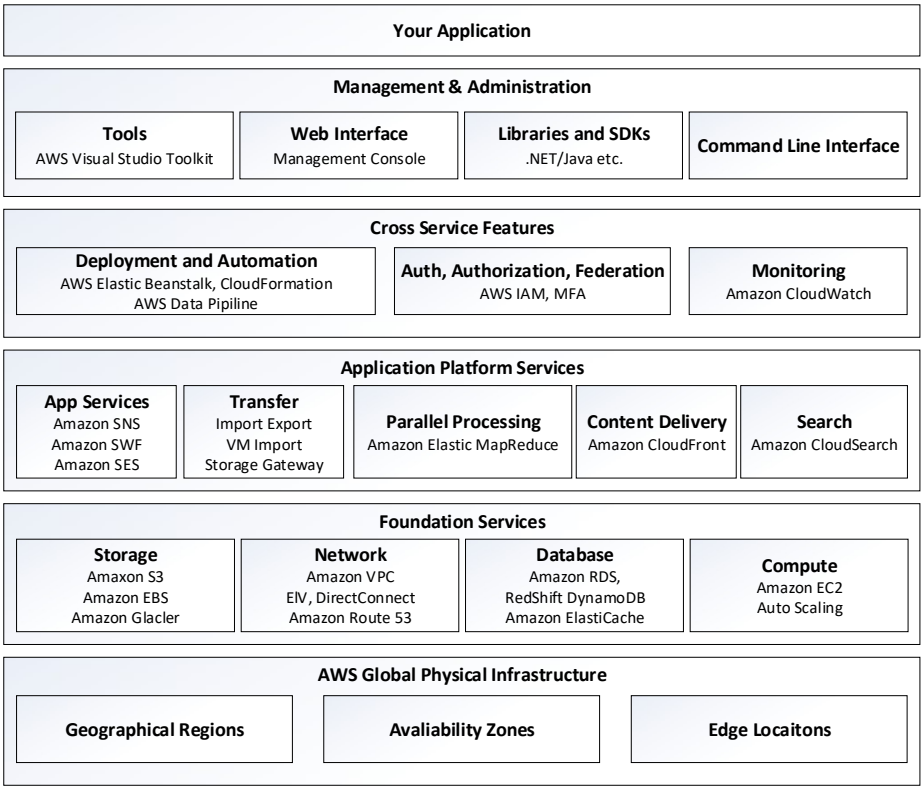


Figure 2. AWS cloud architecture

Amazon Web Services (AWS) [33] is a secure cloud services platform of Amazon.com, offering compute power, database storage, content delivery and other functionality to help businesses scale and grow. Explore how millions of customers are currently leveraging AWS cloud products and solutions to build sophisticated applications with increased flexibility, scalability and reliability. Figure 2 shows the AWS cloud architecture. Many developers choose cloud services of AWS. And in order to make a better choice of cloud services, some developers study and present the characterization of AWS, which is useful for developers aiming at entrusting AWS to deploy their contents [34]. Many other developers focused on the SaaS data protection, they present a real use case of home healthcare SaaS application deployed on AWS,

they also study the challenges needed to add cryptography and key management capabilities to the standard AWS Web/database offer so to enable SaaS data protection [35].

Microsoft Azure [36] is a growing collection of integrated cloud services that developers and IT professionals use to build, deploy, and manage applications through our global network of datacenters. Figure 3 shows the main Azure cloud services architecture. Many developers choose cloud services of Azure. Some developers formulate and evaluate production-feasible methods to develop idleness profiles for customer databases by using Azure SQL Database telemetry across multiple data centers [37]. It is also able to build up a system by combining the Open Plant Communication Universal Architecture and the Microsoft Azure Internet-of-Things Hub [38]. In this paper, we also collect the data by sending messages to Azure IoT Hub.

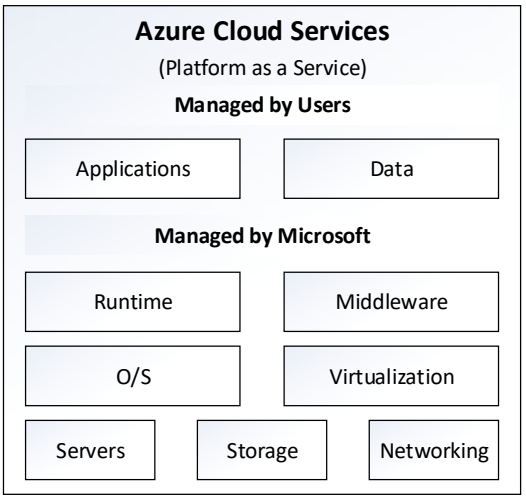


Figure 3. Main cloud service architecture of Azure

Google Cloud Platform [39], offered by Google, provides a set of management tools and a series of modular cloud services including computing, data storage, data analytics and machine learning. Figure 4 shows the Google Cloud Platform services architecture. Many developers choose cloud services of Google Cloud Platform. Some developers use Datastore APIs from Google App Engine (GAE) to interface to different open source distributed database technologies, so the APIs are able to be used by web applications and services without modification [40]. Some other developers present an approach to workload

classification and its application to the Google Cloud Backend [41]. During our simulation experiments, we used the Cloud IoT Core service of Google Cloud Platform.

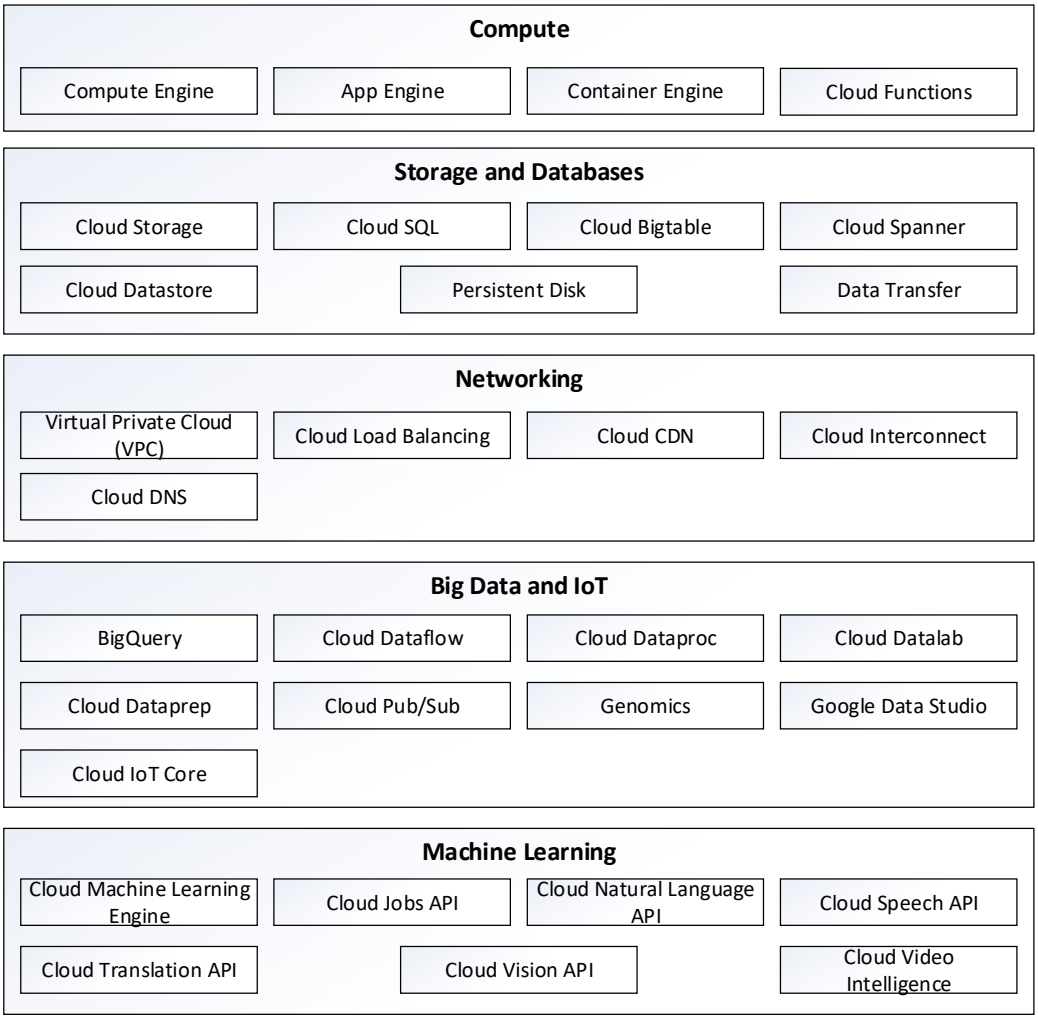


Figure 4. Google Cloud services architecture

Hadoop as an open source project of the Apache foundation is the most representative product for the cloud computing research and application. The Hadoop's distributed framework provides developers with a base architecture for distributed systems. The Hadoop users can develop distributed applications without understanding the underlying details of the distributed system and make full use of the cluster storage resources, network resources and computing resources. The core design of Hadoop is MapReduce and Hadoop Distributed File System (HDFS) [42]. Figure 5 shows the architecture of Hadoop. Hadoop frame

includes four modules: MapReduce, HDFS, YARN and Common Utilities. Hadoop MapReduce is YARN-based system for parallel processing of large data sets. Hadoop Distributed File System (HDFS) is a distributed file system that provides high-throughput access to application data. Hadoop YARN is a framework for job scheduling and cluster resource management. Hadoop Common Utilities are Java libraries and utilities required by other Hadoop modules. These libraries provide filesystem and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.

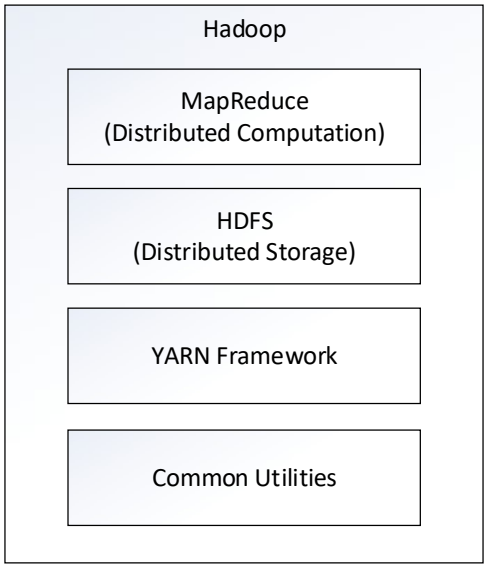


Figure 5. Hadoop architecture.

HDFS is a highly fault-tolerant system, suitable for deployment in cheap machines. HDFS can provide high throughput access data and it is very suitable for large-scale data sets. HDFS has a high fault-tolerance characteristic, and is designed for deployment on low-cost hardware. It provides high throughput to access the application data, suitable for those with large data set applications. HDFS is a distributed file management system for massive data storage. In this system, we use HDFS to store the sensor data files, and we manage the files by calling Hadoop commands in Java Application [43].

2.2 IoT Platforms

The JNU Indoor IoT system is an old project in our lab, which consist of nine modules. Figure 6 shows the conceptual model of the JNU Indoor IoT System. GIS Platform provides geography information service to App Client. Actuator Platform provides actuators' information and actuators' state to App Client. Actuator Middleware connect between Actuator Emulator and Actuator Platform. Actuator Emulator simulate actuators and the state. Sensor Platform provide sensing data and sensors' information to other modules. Sensor Middleware connect between Sensor and Sensor Platform. Sensors collect sensing data and send them to Sensor Middleware. App Server provides services and object information to App Client. App Client show the results to the users via services supported by other modules.

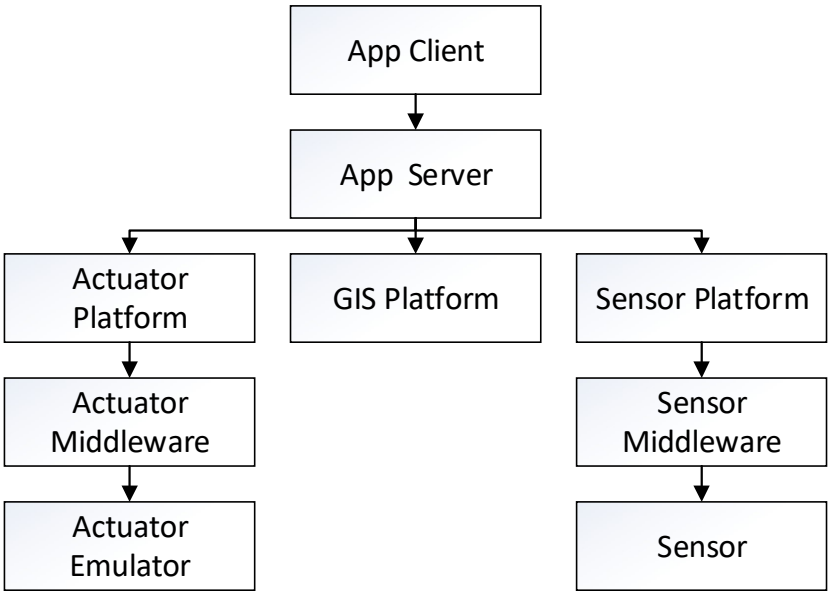


Figure 6. The conceptual model of the JNU Indoor IoT system

2.3 IoT Protocol

REST (Representational State Transfer) [44] is an architectural style, which is often used in the development of web services. REST is a popular building style for cloud-based APIs. A RESTful API means web services used REST architecture. REST architecture involves reading a designated web page that contains an XML file, which describes and includes the needed content. REST typically runs over HTTP (Hypertext Transfer Protocol) and is often used in mobile applications, social networking web sites, mashup tools and automated business processes. REST use a limited number of operations (GET, POST, PUT and DELETE) to enhance the interactions between clients and services. And it is flexible because of assigning resources their own URIs (Universal Resource Identifiers).

CoAP (Constrained Application Protocol) [45] is a software protocol intended to be used in very simple electronics devices, allowing them to communicate interactively over the Internet. It is particularly targeted for small, low-power sensors, switches, valves and similar components that need to be controlled or supervised remotely, through standard Internet networks. CoAP is an application layer protocol that is intended for use in resource-constrained internet devices, such as WSN nodes. CoAP is designed to easily translate to HTTP for simplified integration with the web, while also meeting specialized requirements such as multicast support, very low overhead, and simplicity. Multicast, low overhead, and simplicity are extremely important for Internet of Things (IoT) and Machine-to-Machine (M2M) devices, which tend to be deeply embedded and have much less memory and power supply than traditional internet devices have.

IoTivity [46] is an open source software framework enabling seamless device-to-device connectivity to address the emerging needs of the Internet of Things. The IoTivity is sponsored by the OCF (Open Connectivity Foundation) who is developing a standard specification and certification program to enable the Internet of Things. This open specification is determined to unlock the massive opportunity in the IoT market, accelerate industry innovation and help developers and companies create solutions. The goal of IoTivity is to develop an open source software framework that can seamlessly connect billions of devices

to the future of the Internet world, regardless of operating system and network protocols. One of the most important parts for activating the Internet ecosystem is how can small and medium-sized companies manufacturing various things add an Internet connection function to their products and provide an environment that can easily provide them with smartphone apps or services. IoTivity is a framework for satisfying these requirements, ensuring interoperability between high-quality Internet devices and developing high-speed internet products. It is also expanding the range of open source hardware (e.g. Raspberry Pie, Edison) and software platforms (e.g. Android, iOS, Windows, Linux, etc.) Currently, IoTivity supports Ubuntu, Tizen and Android, and iOS will be supported in the future. The open source hardware platform now supports Arduino and Edison, and will continue to expand its supported hardware platforms. Basically, IoTivity is an open source technology for Internet middleware based on OIC standards.

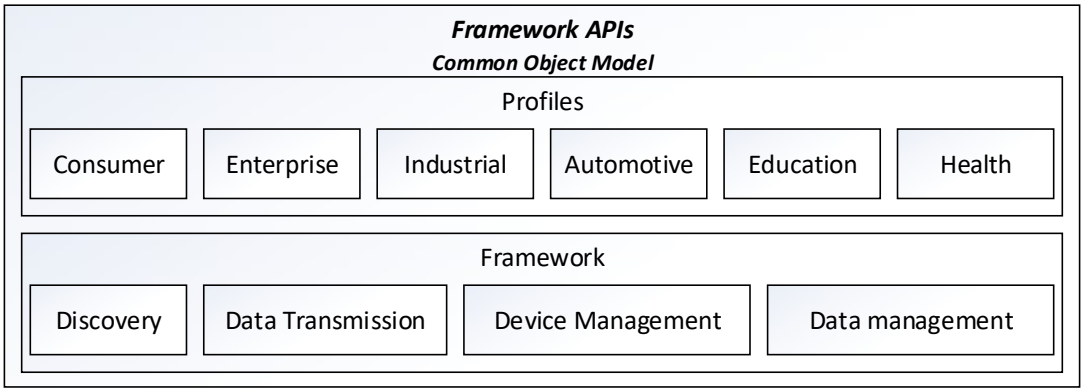


Figure 7. Figure 2.6 OCF IoTivity architecture

Figure 7 shows the conceptual architecture of IoTivity that consists of three layers. Transports layer supports the existing protocols such as Bluetooth, Wi-Fi, Zigbee, etc. Profile layer stands for each vertical field of object Internet applications such as smart home, smart factory, eHealth, etc. A framework layer supports functions such as resource discovery, data transfer, device management, and data management. In the case of the transport layer, new technologies can be continuously extended, and even with these extensions, the application layer of the profile layer can be executed without modification, with the support of the framework layer. For reference, the license policy follows Apache 2.0 and is operated by the Linux

Foundation. IoTivity is based on a resource-based RESTful architecture model, thus representing all things as resources and providing CRUDN (Create, Read, Update, Delete and Notify) operations. In addition, it is designed based on CoAP (Constrained Application Protocol) without a daemon, so it is easy to support low-end and low-power devices.

MQTT [47] is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. For example, it has been used in sensors communicating to a broker via satellite link, over occasional dial-up connections with healthcare providers, and in a range of home automation and small device scenarios. It is also ideal for mobile applications because of its small size, low power usage, minimized data packets, and efficient distribution of information to one or many receivers.

3. CoT Architecture for Hadoop and IoT Platform Using IETF CoAP

3.1 CoT Design for Interworking between Hadoop and IoT Platform

Recently, CoT technology has obtained great development over the last few years and is increasingly influencing various industrial development. Figure 8 shows the interworking architecture between Hadoop and Sensor Platform. As shown in the figure, each Sensor Platform connects with Hadoop. And each sensor platform can connect many sensor middleware, each sensor middleware can connect many sensors. IoT platform provide sensor information and sensing data storage service. Sensor middleware get sensing data from sensors and save the data into database via the service provided by sensor platform. Then IoT platform will request sensing data and sensor information and upload the data to local IoT cloud in multiple sensor networks.

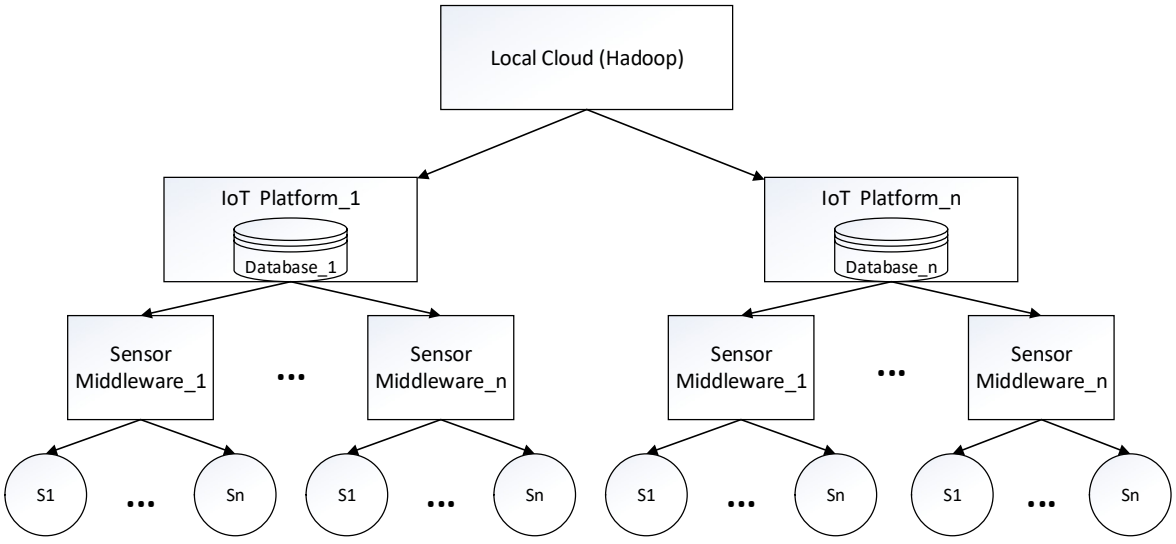


Figure 8. Interworking architecture between IoT platform and local Cloud (Hadoop)

Figure 9 shows the implement architecture for Hadoop and IoT platform. Sensor Middleware will get sensing data from Sensor and save sensing data into database via the service provided by Sensor Platform. There is a RESTful API in Sensor Platform, the communication between IoT platform and Sensor Middleware based on this API and used HTTP protocol. Sensor connect to Sensor Middleware by serial port. The communication between HDFS Agent and IoT platform also based on the RESTful API. There is a HTTP Client in HDFS Agent. HTTP Client will get sensing data and sensor information from IoT platform via the RESTful API. And Data Uploader in HDFS Agent is able to receive the sensing data from HTTP Client, convert the sensing data to files (txt, csv), and upload the files to Hadoop Distributed File System (HDFS) in Hadoop framework. Client will control the process start or stop and show users the sensing results.

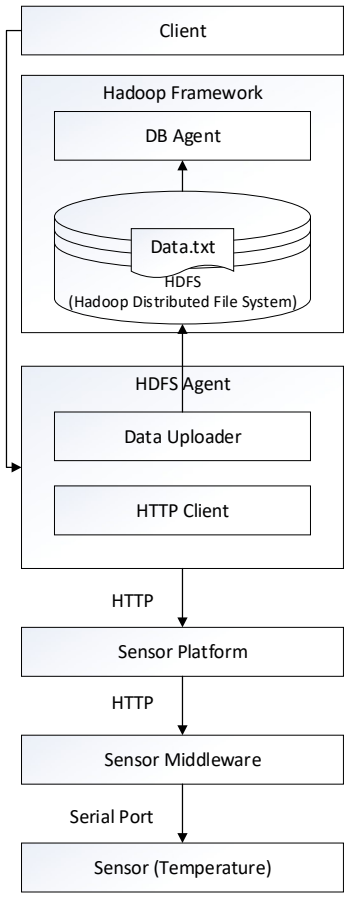


Figure 9. Implement architecture for Hadoop and IoT Platform

Figure 10 shows the detail design for HDFS Agent. In HDFS Agent. There is a HTTP Client and a Data Uploader. HTTP Client is able to request sensing data from HTTP Server in Sensor Platform, Data Uploader is able to upload sensing data file to HDFS. In HTTP Client, there are Data Transmitter, Data Parser and Data Receiver. Data Receiver is able to receive sensing data from HTTP Server, Data Parser is able to convert sensing data to sensing data files, Data Transmitter is able to push the sensing data files to Data Uploader. When HDFS Agent get sensing data from Sensor Platform, Data Receiver in HTTP Client will receive data first and send to Data Parser, then Data Parser will send to Data Transmitter, and Data Transmitter will send data to Data Uploader and Client.

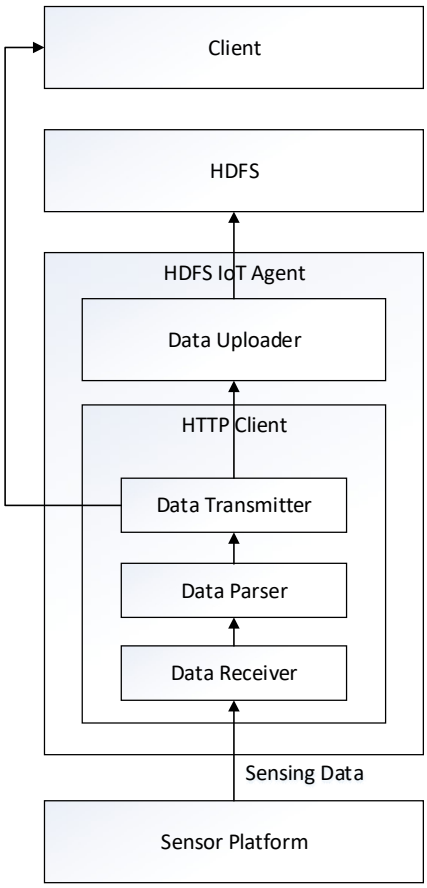


Figure 10. Detail design for HDFS Agent.

Figure 11 shows the detail design for Sensor Platform. Service Interface provides access interface to outside service. The main services offered by Sensor Web module are Sensor Web Content Service, Sensor

Web Provider Service and Sensor-Web Sensing Service. Sensor Web Content Service is used for middleware configuration Management and Sensor Information Management. Sensor Web Provider is utilized for Sensor Searches, Sensor Information supply and Sensing Data supply. Sensor Web Sensing Service is used for Sensor State management and as Sensing Data receiver.

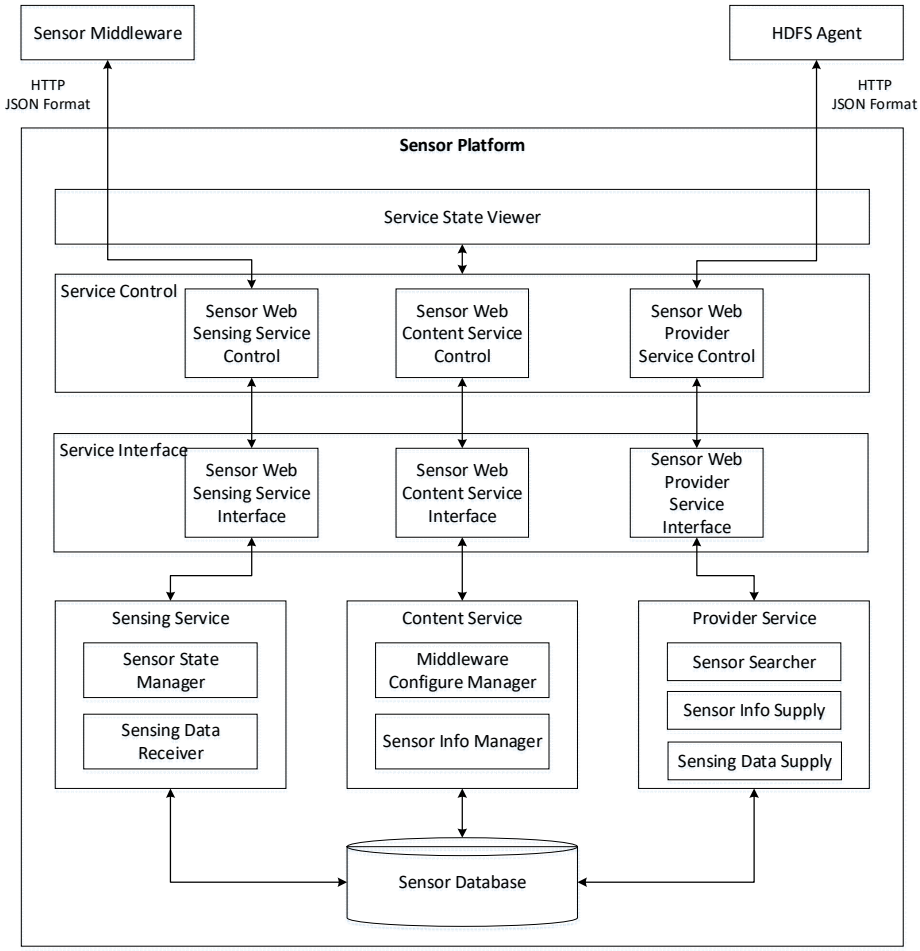


Figure 11. Detail design for Sensor Platform.

Figure 12 shows the detail design for Sensor Middleware. Sensor middleware take a role in collecting sensing data sent by sensor then sending and saving Sensor Platform. Configurator request sensor information (ID, Type) from Sensor Platform, and also verify pertinent IP address and Platform access privileges. Sensing Driver takes various sensors' sensing data format information and parse processing through received sensing data. Port Monitoring take a role in monitoring state of the port connected with

middleware. Sensing Data Receiver accesses sensing data sent from sensor node and saves at memory through sensing data Parser. Sensing Data Transporter read Memory-saved sensing data and send to Sensor Platform.

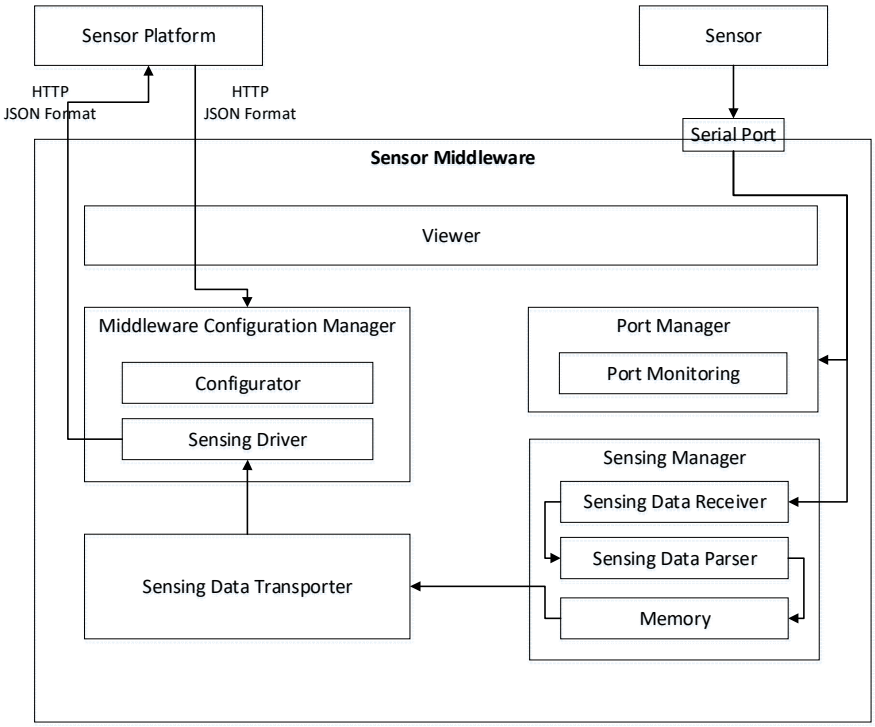


Figure 12. Detail design for Sensor Middleware.

Figure 13 shows the sensor which connect to Sensor Middleware via Serial Port.

Figure 13. Temperature sensor connect with Sensor Middleware

Figure 14 shows the sequence diagram for the interworking between Hadoop and IoT Platform. Client send start request to Data Uploader and request sensing data from Data Uploader. Data Uploader sends request message to HTTP Client, HTTP Client sends request message to sensor Platform, IoT platform sends request message to Sensor Middleware, Sensor Middleware sends request to Sensor and Sensor will return sensing data to Sensor Middleware. Sensor Middleware will return sensing data to Sensor Platform, IoT platform will save sensing data into database and return sensing data to HTTP Client. HTTP Client will return sensing data to Data Uploader. Data Uploader will return sensing data to Client, make a data file (txt, csv) and uploads file to HDFS. Finally, Client will send stop request to Data Uploader, all process will stop.

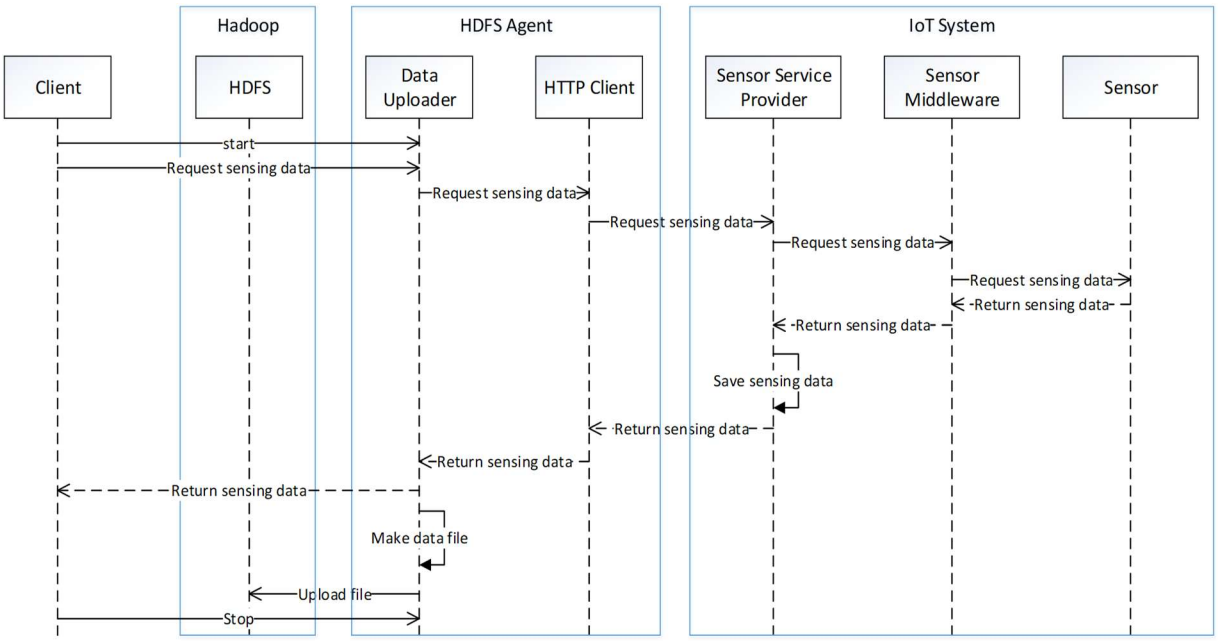


Figure 14. Sequence diagram of this system for the interworking between Hadoop and IoT Platform

3.2 Experiments and Results of CoT Architecture for Hadoop and IoT Platform

Table 1 shows the development environment for the IoT Platform

Components	Version
Operating System	Windows 10
Microsoft Visual Studio	2015
Microsoft SQL Server	2016

Table 1. IoT Platform development environment.

Table 2 shows the development environment for HDFS Agent.

Components	Version
Operating System	Windows 10
Java	JRE 1.8
Spring Tool Suite	3.8.4

Table 2. HDFS agent development environment.

Table 3 shows the configuration environment for Hadoop.

Components	Version
Operating System	Windows 10
Java	JRE 1.8
Hadoop	2.7.3
Sprint Tool Suite	3.8.4

Table 3. Hadoop configuration environment.

To run the whole system, we need to run IoT platform firstly, the result shows in figure 15. “Service State Viewer” shows the state of services. “Time Now” shows the current time. The first “Run Time” shows the running time of Provider Service and “Provider Service” shows the state of Provider Service (start or stop). The second “Run Time” shows the running time of Content Service and “Content Service” shows the state of Content Service (start or stop). “Node Count” shows the number of sensors. “Provider Service Control” has two buttons to control Provider Service and Sensing Service start or stop. “Content Service Control” has two buttons to control Content Service start or stop. “Service Load Viewer” shows the control history and click “Clear Control History” button will clear the control history.

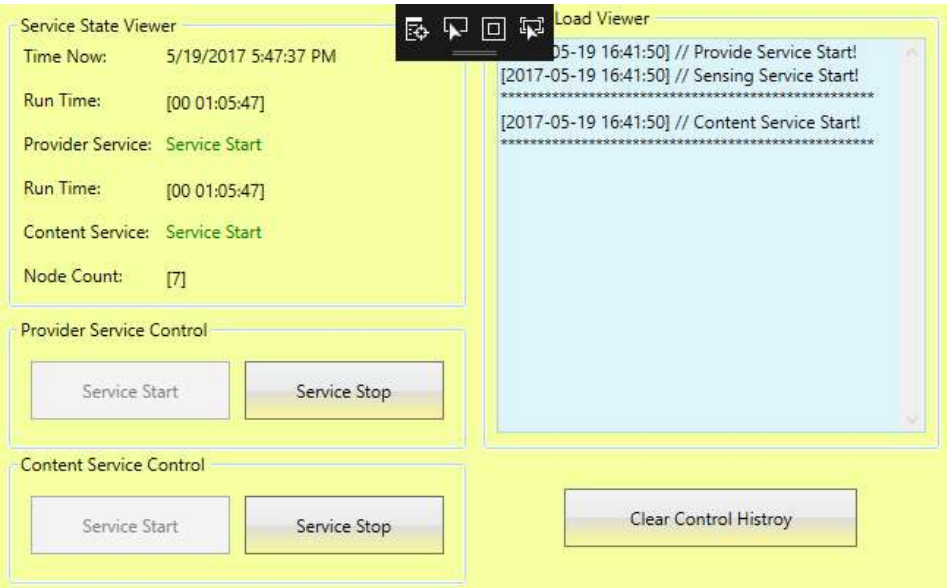


Figure 15. IoT platform implementation result.

Then we need to run Sensor Middleware, the result shows in figure 16. "Sensing Data" shows the sensor information and sensing data. Users can choose a sensor in "Sensor ID" and the first textbox will show the information of the choose sensor. User can also choose the serial port for temperature sensor, humidity sensor and illumination sensor on the right. Click "Start" button to start the connection and send sensing data to IoT platform to save data into Database. Click “Stop” button to stop the connection. The second textbox will show the sensing data.

Finally, we need to run Client, the result shows in figure 17. We need to choose a sensor in the combo box, if the sensor is not in using, we cannot click the “start” button to start working. After we choose a sensor, the sensor information text area will show the information of this sensor. Click “start” button will get sensing data through Sensor Platform, make data files and upload files to HDFS. Click “Stop” button will stop all the process.

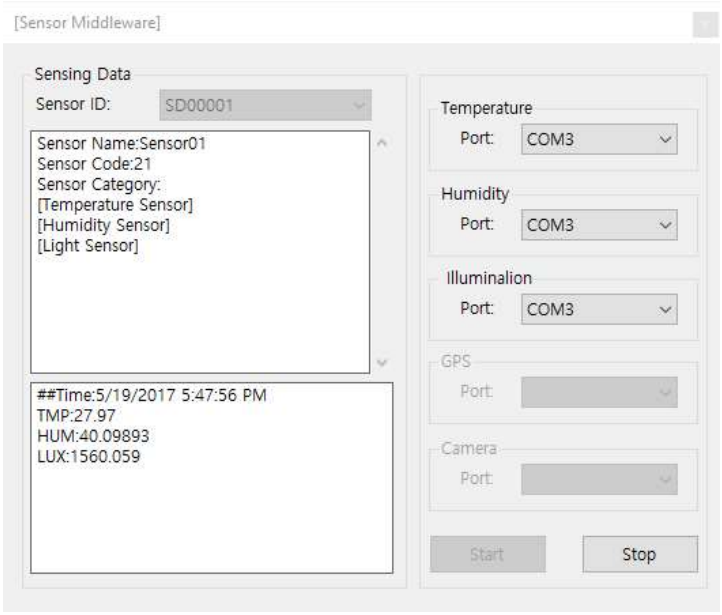


Figure 16. Sensor Middleware implementation result.

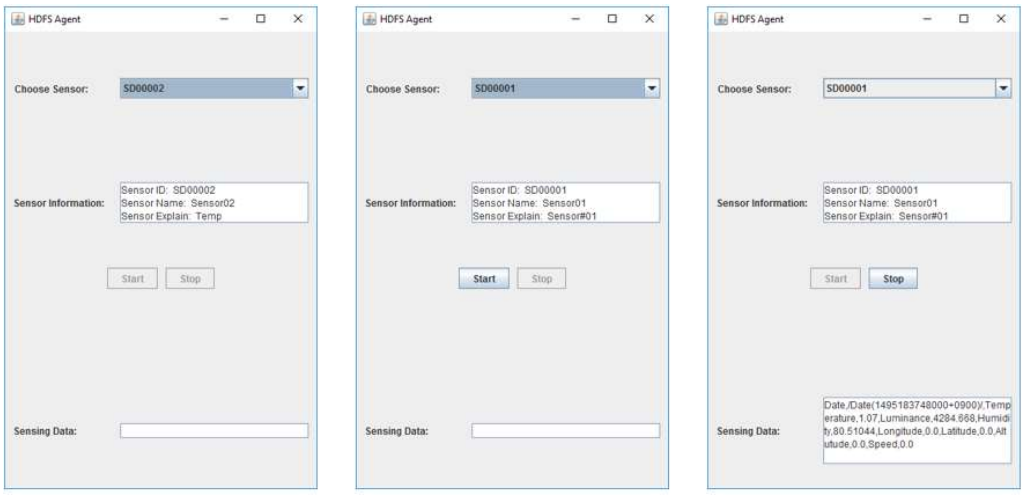


Figure 17. Client implementation result.

After finish running the whole system, we can check the files storage situation in our local file system, like the figure 18 shows.

And we can also check the files storage situation in Hadoop Distributed File System, like the figure 19 shows.

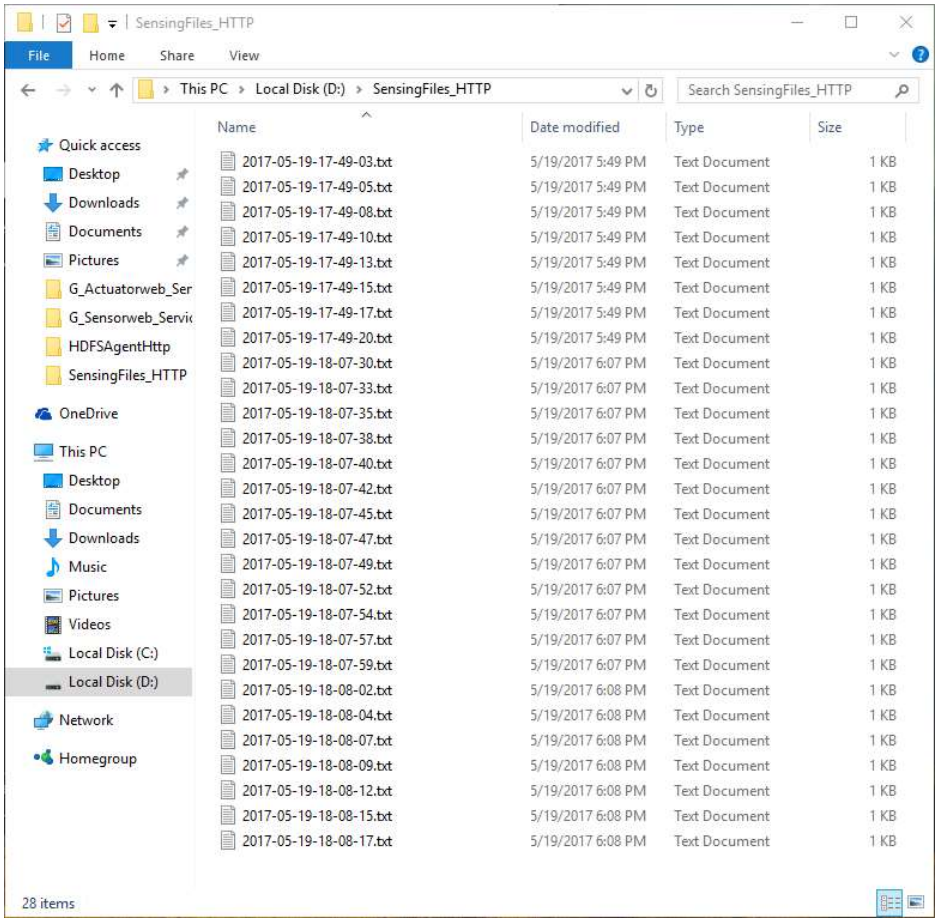


Figure 18. Sensing file list in local file system.

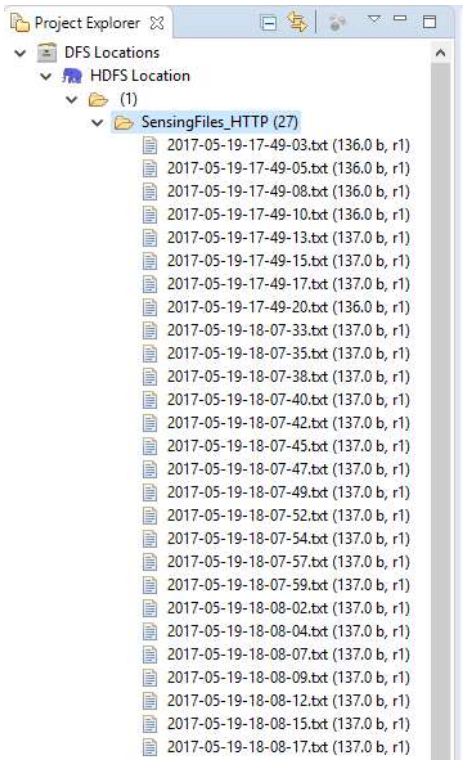


Figure 19. Sensing file list in Hadoop Distributed File System.

In file list, each file's name is current time of sensing data. And the content is a string split by commas including sensing time and sensing data. Figure 20 shows the sensing data on 17:49:03, May 19, 2017.

Figure 20. A file of the sensing list in HDFS.

4. CoT Architecture for Hadoop and IoT Networks Based on Agent Using IETF CoAP

4.1 CoT Design for Interworking between Hadoop and IoT Networks

Recently, it is increasing to research the interworking between Cloud and IoT Devices. Figure 21 shows the interworking conceptual model of Hadoop and IoT Networks. As shown in the figure, Hadoop is able to connect many agents, each agent can connect many IoT devices and each IoT device can connect many sensors. IoT Device will get sensing data from sensors and send to the agent, and agents will upload sensing data to Local Cloud.

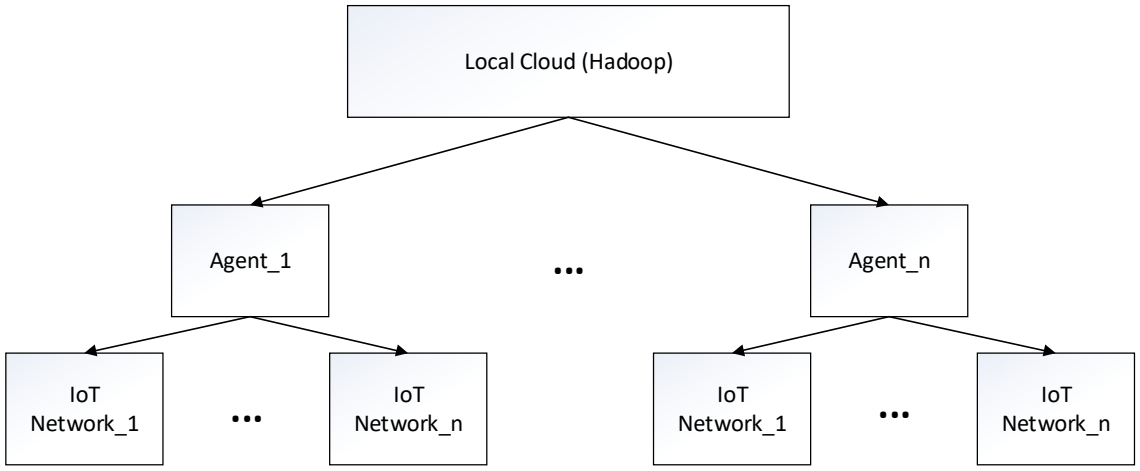


Figure 21. Conceptual model of the architecture for Hadoop and IoT Networks

Figure 22 shows the interworking layer of Hadoop and IoT Networks. We will connect a temperature sensor to Edison Board, and develop a CoAP Server in Edison Board which can get sensing data from sensor and provide data to CoAP Client. And we will develop a HDFS Agent, which consist of two parts:

Data Uploader and CoAP Client. Data Uploader is able to receive the sensing data from CoAP Client, convert these data to files (txt, csv) and upload files to Hadoop Distributed File System.

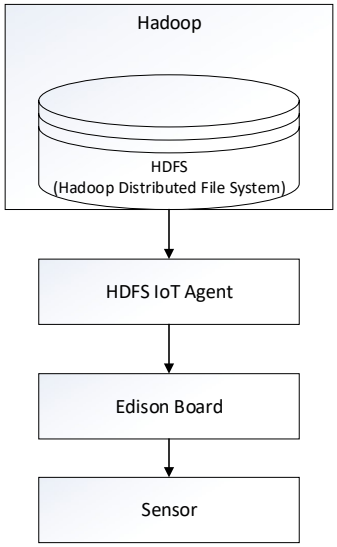


Figure 22. Interworking layer of Hadoop and IoT Networks

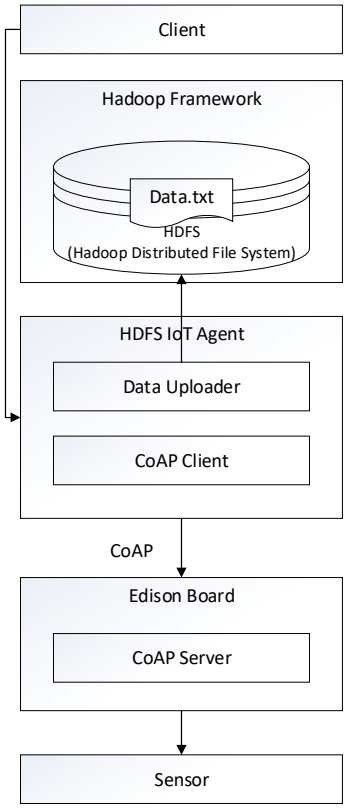


Figure 23. Implement architecture for Hadoop and IoT device

Figure 23 shows the implement architecture of for Hadoop and IoT device. There is a CoAP Server in Edison Board, which can get sending data from sensor and send to CoAP Client in HDFS Agent when CoAP Client sends request. Data Uploader in HDFS Agent is able to receive the sensing data, make data files (txt, csv) and upload the file to Hadoop Distributed File System (HDFS) in Hadoop framework. And Client will control the process start or stop and show users the sensing results.

Figure 24 shows the design for HDFS IoT Agent. In HDFS IoT Agent. There is a CoAP Client and a Data Uploader. In CoAP Client, there are Data Transmitter, Data Parser and Data Receiver. When HDFS IoT Agent get sensing data from Edison Board, Data Receiver in CoAP Client will receive data first and send to Data Parser, then Data Parser will send to Data Transmitter, and Data Transmitter will send data to Data Uploader and Client.

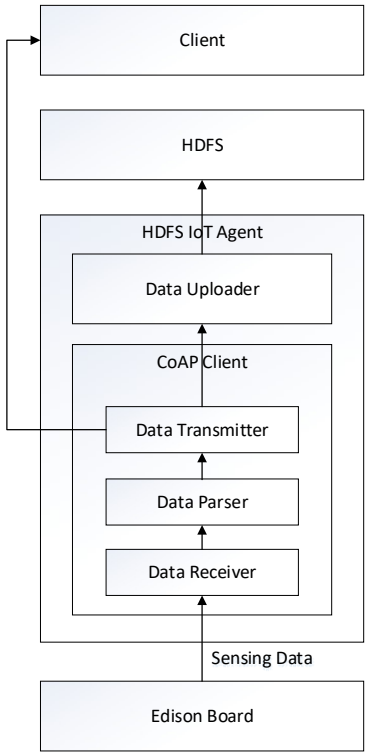


Figure 24. The design for HDFS IoT Agent.

Figure 25 shows the sequence diagram for the interworking between Hadoop and IoT device. Client send start request to Data Uploader and request sensing data from Data Uploader. Data Uploader sends request

message to CoAP Client, CoAP Client sends request message to CoAP Server, and CoAP Server requests sensing data from sensor. After getting sensing data from sensor, CoAP Server will return sensing data to CoAP Client, CoAP Client will return sensing data to Data Uploader, Data Uploader will return sensing data to Client. And Data Uploader will make a sensing data file (txt, csv) and upload the file to HDFS (Hadoop Distributed File System). Finally, Client send stop request to Data Uploader, all process will stop.

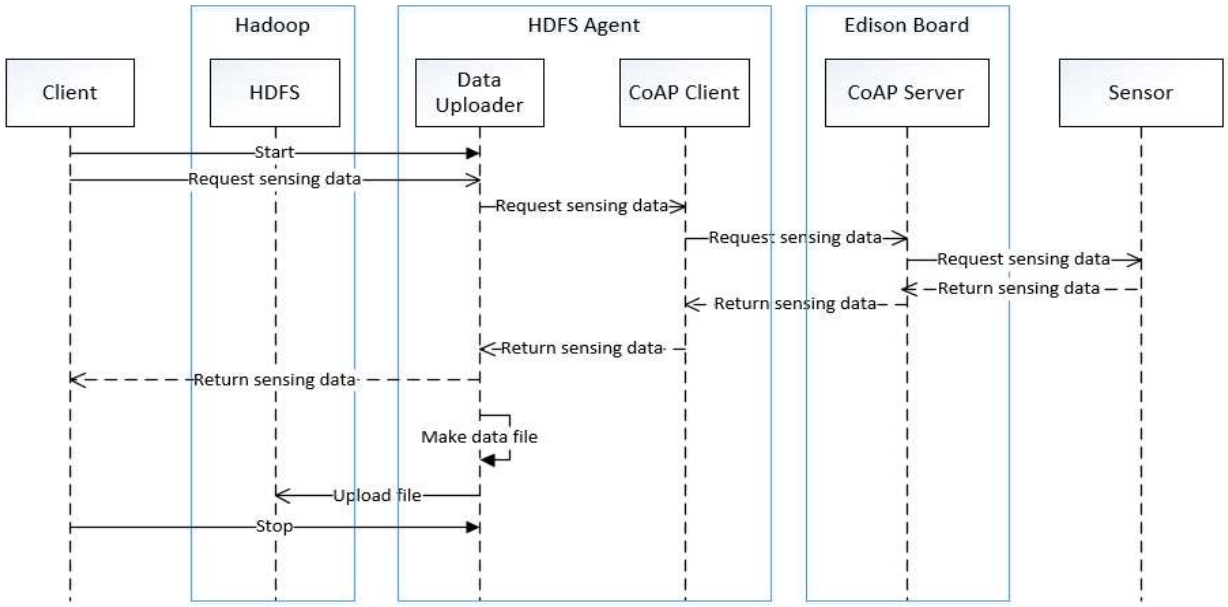


Figure 25. Sequence diagram for the interworking between Hadoop and IoT device

4.2 Experiments and Results of CoT Architecture for Hadoop and IoT Networks

Table 4 shows the development environment for CoAP Server in Edison Board.

Components	Version
Operating System	Windows 10
Edison Board (Yocto)	20160606
Intel System Studio IoT Edison	2016.4.012

Table 4. CoAP Server development environment.

Table 5 shows the development environment for HDFS Agent.

Components	Version
Operating System	Windows 10
Java	JRE 1.8
Spring Tool Suite	3.8.4

Table 5. HDFS Agent development environment.

Table 6 shows the configuration environment for Hadoop.

Components	Version
Operating System	Windows 10
Java	JRE 1.8
Hadoop	2.7.3
Sprint Tool Suite	3.8.4

Table 6. Hadoop configuration environment.

Figure 26 shows the Edison Board and temperature sensor.

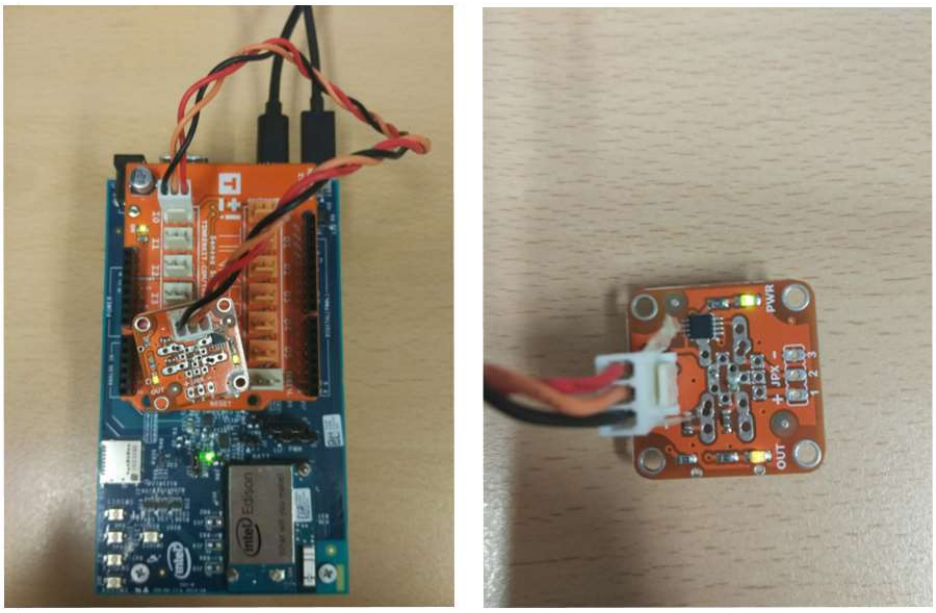


Figure 26. Edison Board and temperature sensor.

Figure 27 shows the CoAP Server implementation result. CoAP Server is run in Edison Board using C language and Intel System Studio IoT Edison, each time CoAP Client sends a request, CoAP Server will return and print out real-time temperature data.

Figure 27. CoAP Server implementation result.

Figure 28 shows the Client implementation result. Click “Start” will get sensing data from CoAP Server in Edison Board and make data files to upload to Hadoop Distributed File System. Click “Stop” button will stop all the process.

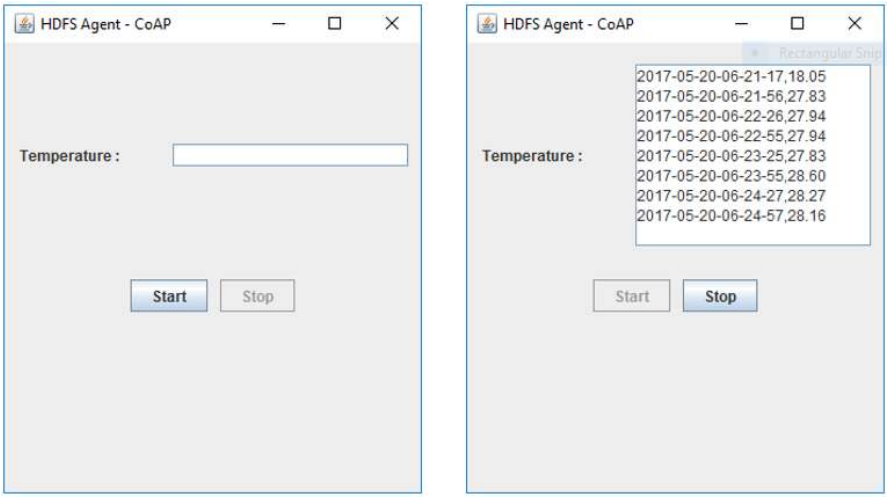


Figure 28. Client implementation result.

After finish running the whole system, we can check the files storage situation in our local file system, like the figure 29 shows.

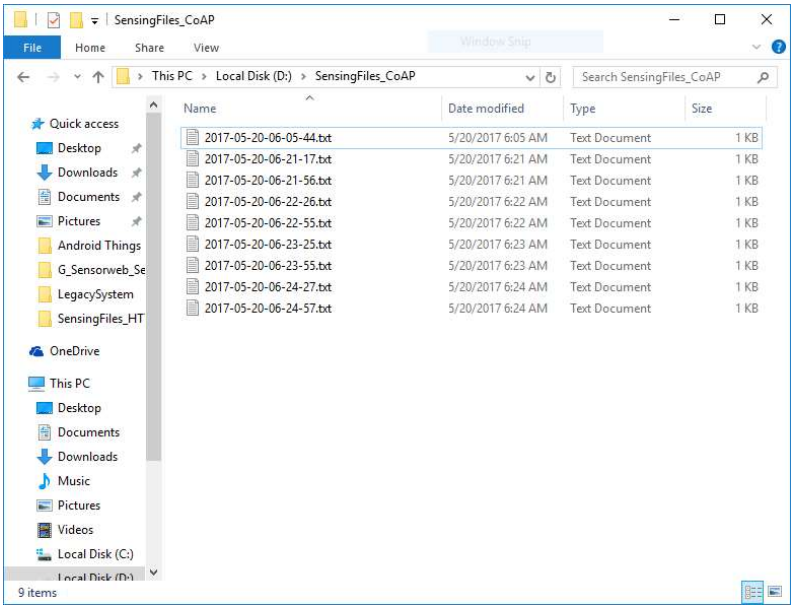


Figure 29. Sensing file list in local file system.

And we can also check the files storage situation in Hadoop Distributed File System, like the figure 30 shows.

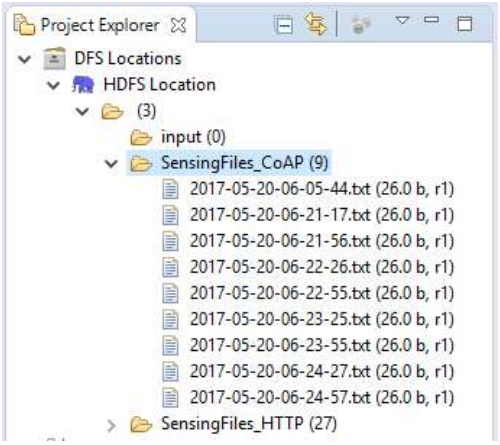


Figure 30. Sensing file list in Hadoop Distributed File System.

In file list, each file's name is current time of sensing data. And the content is a string split by a comma including sensing time and sensing data. Figure 31 shows the temperature is 17.70°C on 6:59:44, May 20, 2017.

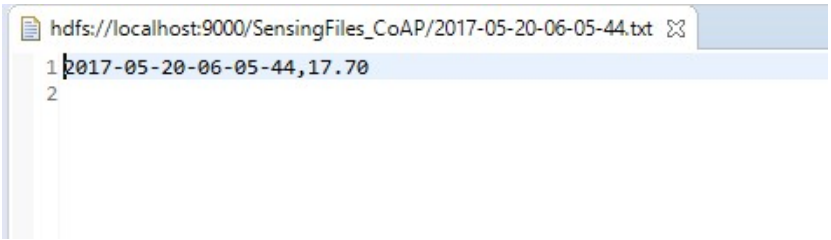


Figure 31. A file of the sensing list in HDFS.

5. CoT Architecture for Global Cloud and IoT Networks Using MQTT

5.1 CoT Design for Interworking between Global Cloud and IoT Networks

Recently, Clouds are wildly used for huge data repository and Internet services in various fields. And IoT networks collect a context data and support the monitor-ing and control services using thing virtualization. We will build the connection be-tween IoT and cloud, it is very useful, and supports intelligent services based on huge context data. This paper presents the comparison analysis of IoT services based on Clouds for huge context acquisition in large scale IoT networks. And, we develop AWS, Azure, and Google cloud based on IoT. And compare the IoT service of AWS, Azure, and Google Cloud by sending sensing data messages from IoT devices.

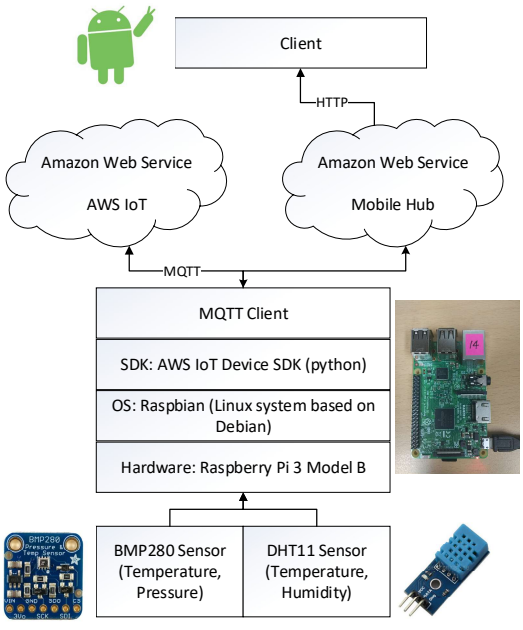


Figure 32. The configuration for the connection between IoT device and AWS

Figure 32 shows the detail architecture for the connection between IoT device and AWS. We used Raspberry Pi 3 Model B with Raspbian system as IoT device, we also used DHT11 Sensor (Temperature, Humidity) and used BMP280 Sensor (Temperature, Pressure). We installed AWS IoT Device SDK with python, and the device-to-cloud messages are send based on MQTT protocol. The MQTT messages in AWS cannot store automatically, so before the device publish message to the topic in AWS, we will create the message item in NoSQL Database of AWS Mobile Hub to store the sensing messages. Client is able to get message item from NoSQL Database of AWS Mobile Hub.

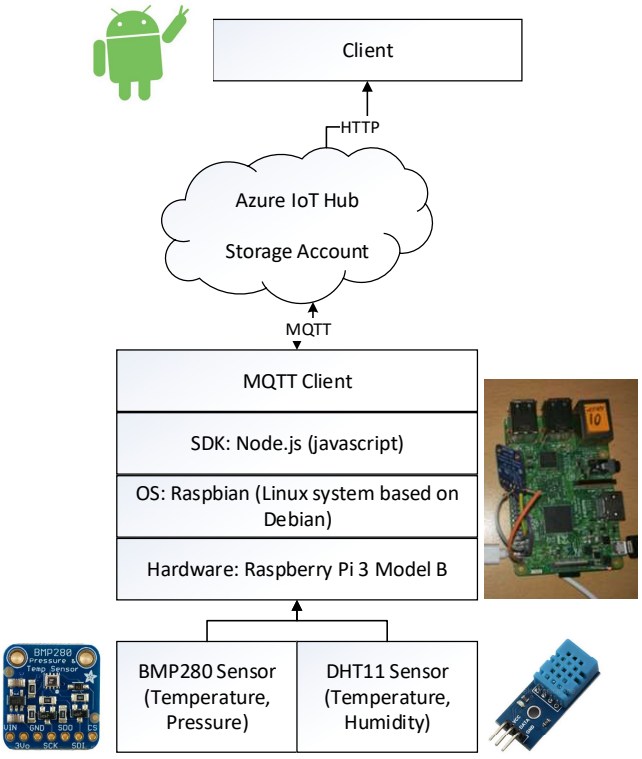


Figure 33. The configuration for the connection between IoT device and Azure IoT Hub

Figure 33 shows the detail architecture for the connection between IoT device and Azure IoT Hub. We used Raspberry Pi 3 Model B with Raspbian system as IoT device, we also used DHT11 Sensor (Temperature, Humidity) and used BMP280 Sensor (Temperature, Pressure). We installed Azure IoT Hub SDK with JavaScript, and the device-to-cloud messages are sent based on MQTT protocol. After the

messages published to the topic of Azure, the messages will be stored in the Blobs of Storage Account of Azure, and Client is able to get message files from Blobs of Azure.

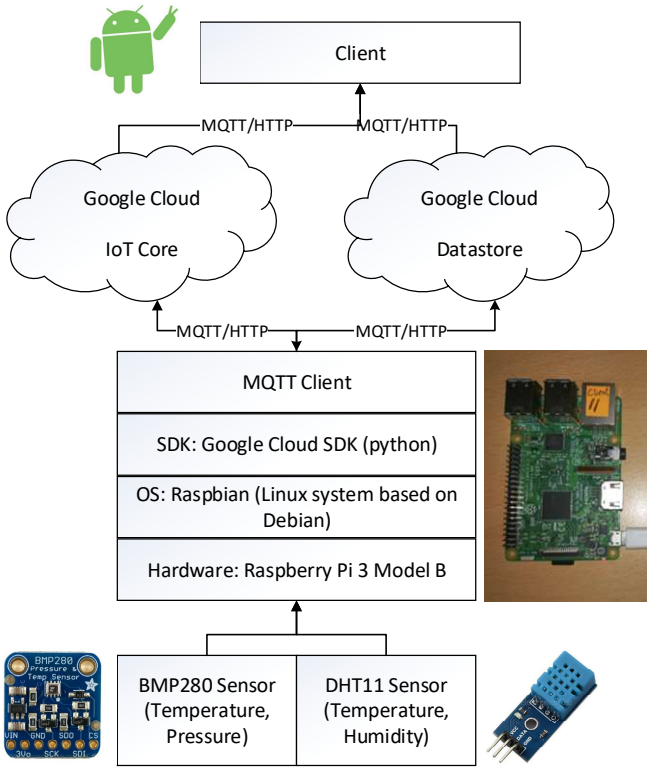


Figure 34. The configuration for the connection between IoT device and Google Cloud

Figure 34 shows the detail architecture for the connection to Google Cloud IoT Core. We used Raspberry Pi 3 Model B with Raspbian system as an IoT device, we also used DHT11 Sensor (Temperature, Humidity) and used BMP280 Sensor (Temperature, Pressure). We installed Google Cloud SDK with python, and the device-to-cloud messages are sent based on MQTT protocol. The messages published to topic in Google Cloud will disappear after several days, so if we want to store the messages, we need to use Google Cloud Datastore. When we publish the message to the topic in Google Cloud, we also create a message entity in Google Cloud Datastore. Client is able to pull message from subscriptions in Google Cloud, if the message has been disappeared, Client is able to get message entity from Google Cloud Datastore.

5.2 Experiments and Results of CoT Architecture for Global Cloud and IoT Networks

Table 7 shows the simulation experiment environment of the CoT architecture for Global cloud and IoT Networks. In each Raspberry Pi, we use Raspbian for IoT device and the cloud SDKs. For the SDK, we used AWS IoT Device SDK with python, Azure SDK with JavaScript, and Google Cloud SDK with python. All of the communication method is MQTT protocol, we used Python 2 to compile the code and added DHT11 driver library and Paho MQTT Client library. And for our experiment, we considered the published message on AWS are able to showed more directly, the published message on Azure need to be download, and the published message on Google Cloud should show by the shell command.

	AWS	Azure	Google Cloud
IoT device	Raspbian	Raspbian	Raspbian
SDK	AWS IoT Device SDK (python)	Node.js (JavaScript)	Google Cloud SDK (python)
Software	Python 2	Python 2	Python 2
Libraries	DHT11 driver, Paho MQTT Client	DHT11 driver, Paho MQTT Client	DHT11 driver, Paho MQTT Client
Webpage Result	More directly	Download	Shell command

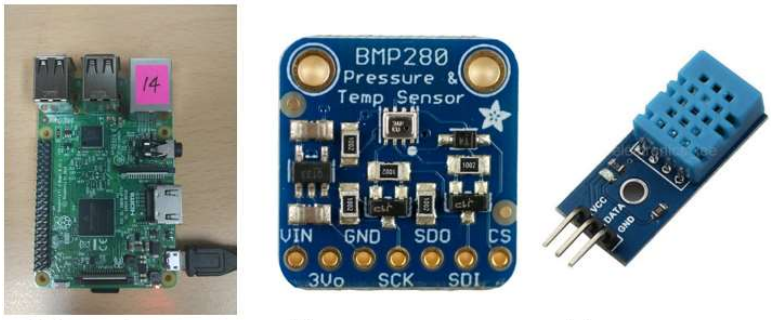
Table 7. Experiment environment of the CoT architecture for Global cloud and IoT Networks

Table 8 shows the development environment of mobile client. We used Android Studio 3.0.1, Android SDK 27, AWS Java SDK 1.11.293, Azure IoT SDK 1.7.23, Google Cloud PubSub 0.45.0, Google Cloud Storage 1.26.0, and Google Cloud Datastore 1.31.0.

Components	Version
Android Studio	3.0.1
Android SDK	27
AWS Android SDK	2.6
Azure IoT SDK	1.7.23
Google Cloud PubSub	0.45.0
Google Cloud Storage	1.26.0
Google Cloud Datastore	1.31.0

Table 8. Development environment of mobile client

Figure 35 shows the IoT device and sensors we used in the CoT architecture for Global cloud and IoT Networks. We used Raspberry Pi 3 Model B with Raspbian system as IoT devices, which shows in figure (a). We also used DHT11 Sensor (Temperature, Humidity) like figure (b) and BME280 Sensor (Temperature, Humidity, Pressure) like figure (c).



(a) Raspberry Pi (b) BMP 280 Sensor (c) DHT11 Sensor

Figure 35. IoT device and sensors used in the CoT architecture for Global cloud and IoT Networks

Figure 36 shows the AWS IoT webpage, it's able to create a MQTT client and enter the same topic, which is able to show the published messages. Figure 37 shows one of the published messages, the format of this message is JSON.

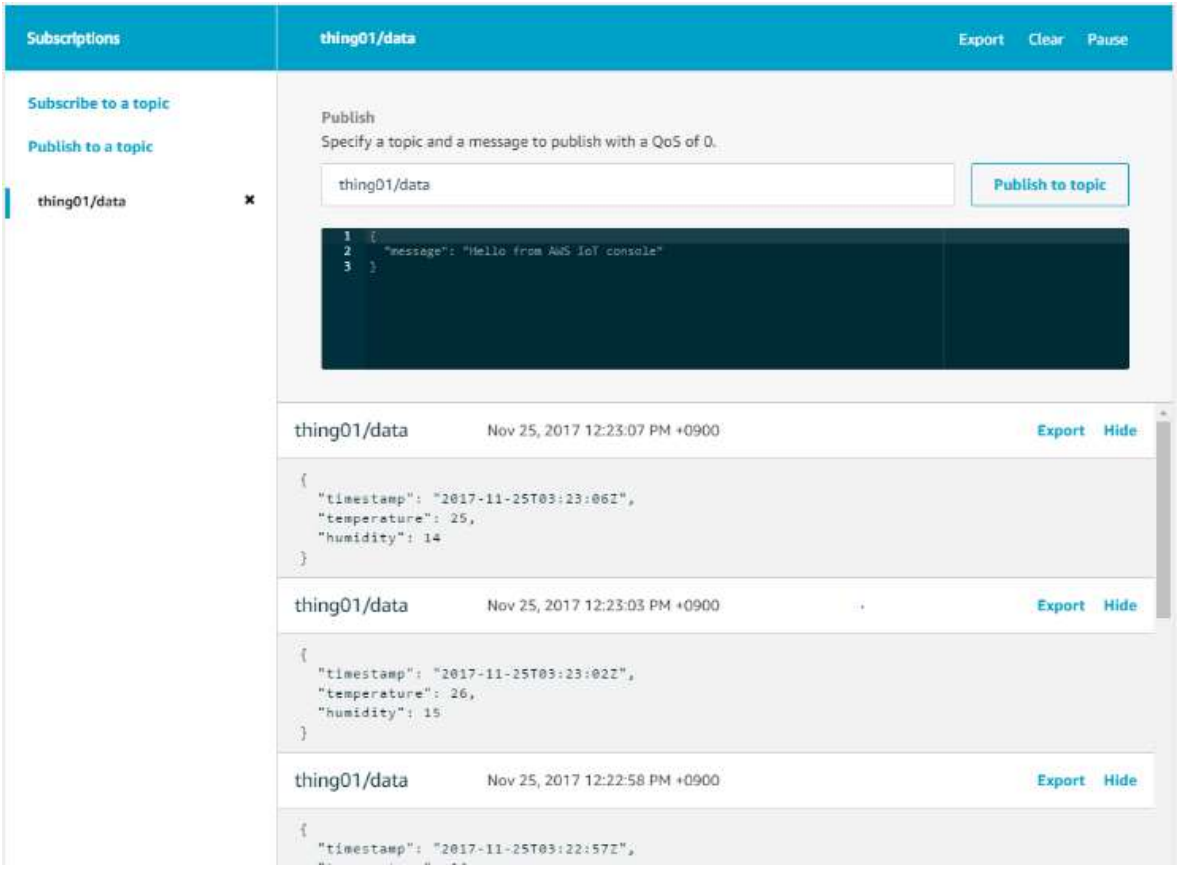


Figure 36. Published messages in topic of AWS

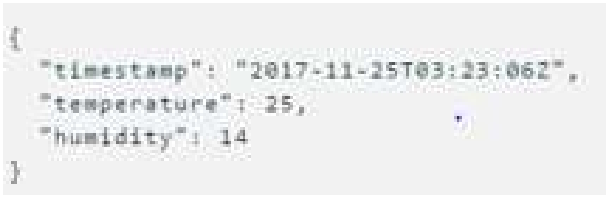


Figure 37. Context messaged published in AWS IoT topic

Figure 38 shows the successful connections with AWS, When the connections become stable, the message publish speed is 0.25/s, the maximum speed is 0.25/s, the minimum speed is 0/s.

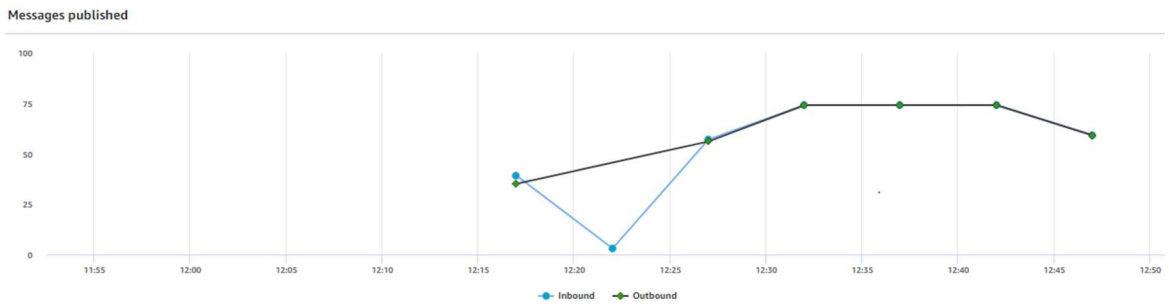


Figure 38. The successful connections with AWS

Figure 39 shows the data stored in NoSQL database of AWS Mobile Hub. The name of the table is finaltest-mobilehub-887317787-messages, the primary sort key is the date, each item includes date, temperature value, and pressure value. For example, as shown in figure, there is an item shows information include date (2018-05-31-07:11), pressure value (975.37616), and temperature value (23.746326).

Scan: [Table] finaltest-mobilehub-887317787-messages: ... Viewing 1 to 100 items

Scan | [Table] finaltest-mobilehub-887317787-messages: userId, date

Add filter

Start search

userId	date	pressure	temperature
raspberry-1	2018-05-31-07:17	975.3861	23.777096
raspberry-1	2018-05-31-07:18	975.3786	23.720682
raspberry-1	2018-05-31-07:19	975.40607	23.720682
raspberry-1	2018-05-31-07:20	975.3562	23.638622
raspberry-1	2018-05-31-07:21	975.37854	23.76684
raspberry-1	2018-05-31-07:22	975.46857	23.674522
raspberry-1	2018-05-31-07:23	975.59595	23.469368
raspberry-1	2018-05-31-07:24	975.5835	23.664265
raspberry-1	2018-05-31-07:25	975.40594	23.895054
raspberry-1	2018-05-31-07:26	975.32837	23.930952

Figure 39. Data stored in AWS NoSQL database

Figure 40 shows the received message file in Azure, it's not able to show messages directly, we need to download the message file. In the figure, there is the Blob service of the storage we created, it is able to find the messages folder.

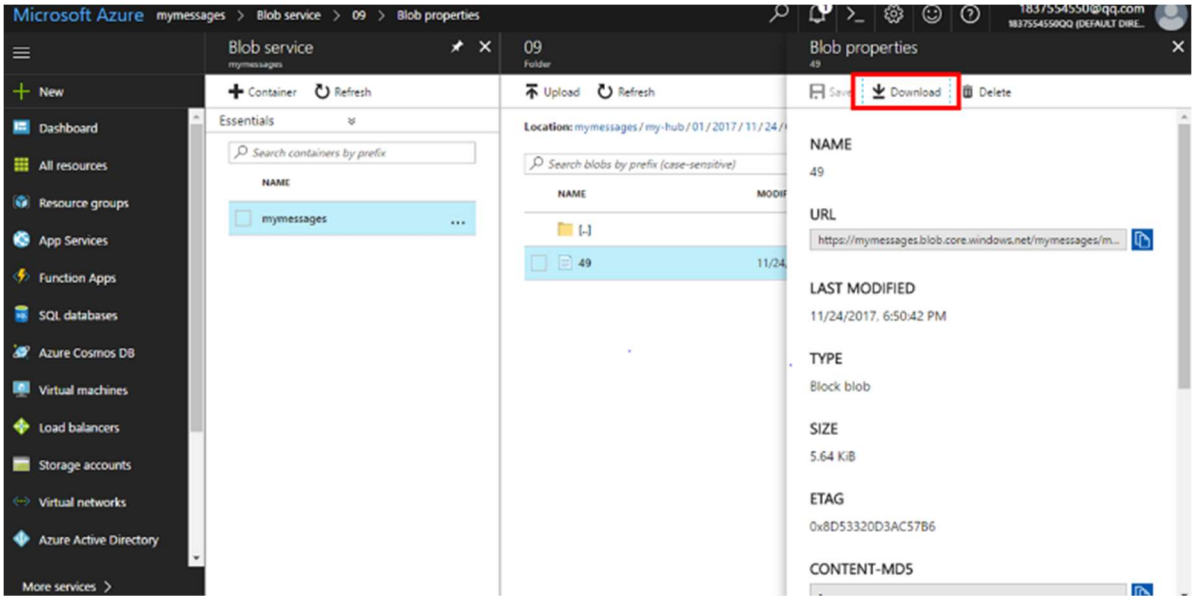


Figure 40. Developed results in Azure IoT Hub

Figure 41 shows the downloaded message file from Azure, it is able to find the sensing data in the file. For example, as shown in the figure, there is the information include the message ID, the correlation ID, the device ID, Time (2018-02-09-11:59:41), the temperature value (24.16173), and the pressure value (807.6015).

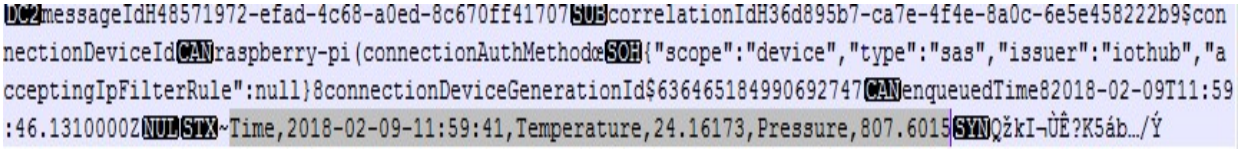


Figure 41. Context message download from Azure IoT Hub

Figure 42 shows the sum messages delivered to storage endpoints of Azure. When the connections become stable, the message publish speed is 0.55/s, the maximum speed is 0.55/s, the minimum speed is 0/s.

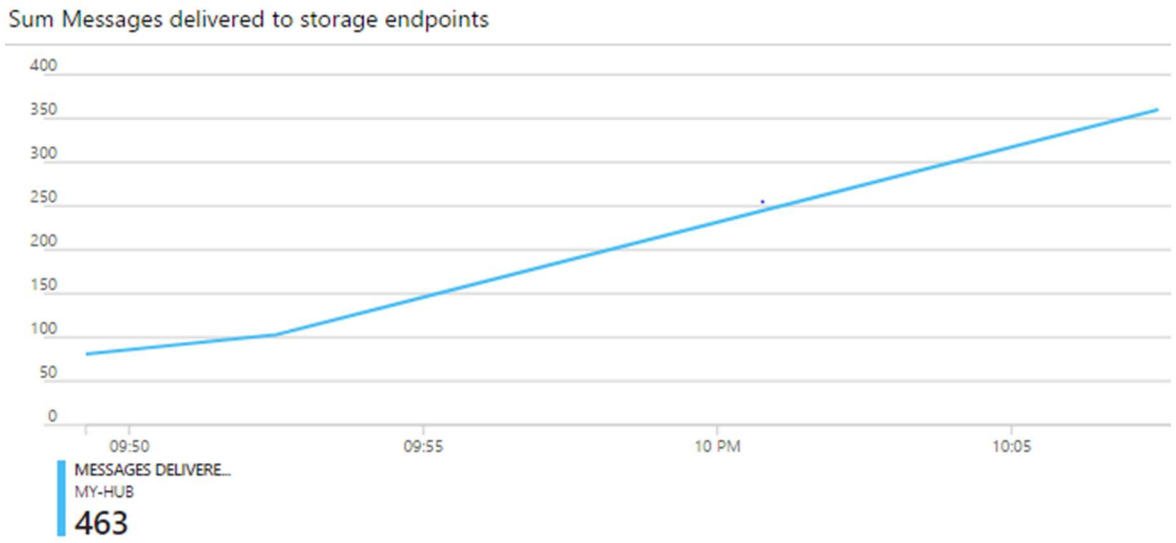


Figure 42. Sum messages delivered to storage endpoints of Azure

Figure 43 shows the received messages on Google Cloud Platform, we need google cloud shell command to check the message. And as shown in the figure, it is able to find the information include the message ID, the device ID, the device registry ID, the device registry location, the project ID, the temperature value (23.377046585083008), and the pressure value (983.5136108398438). (The detailed message shows in figure 44)

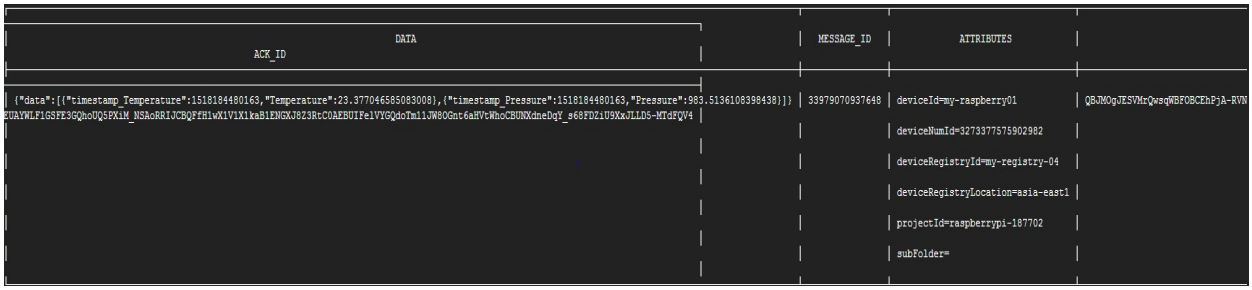


Figure 43. Context message published in Google Cloud topic

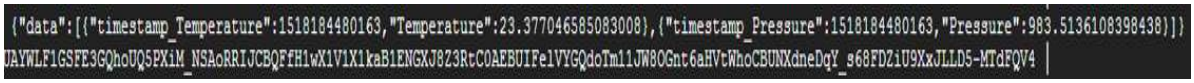


Figure 44. Context message published in Google Cloud topic (detail)

Figure 45 shows the publish message operations to the topic in Google Cloud Platform. When the connections become stable, the message publish speed is 0.81/s, the maximum speed is 0.95/s, the minimum speed is 0/s.

Figure 45. The publish message operations to the topic in Google Cloud Platform

Kind

messages

Filter entities

Name/ID	date	pressure	temperature
name=2018-06-01-13:05	2018-06-01-13:05	979.9617	27.898651
name=2018-06-01-13:06	2018-06-01-13:06	979.9559	27.96526
name=2018-06-01-13:07	2018-06-01-13:07	979.99164	27.924267
name=2018-06-01-13:08	2018-06-01-13:08	979.99194	27.66294
name=2018-06-01-13:09	2018-06-01-13:09	980.0554	27.632196
name=2018-06-01-13:10	2018-06-01-13:10	979.93085	27.744925
name=2018-06-01-13:11	2018-06-01-13:11	979.9617	27.898651
name=2018-06-01-13:12	2018-06-01-13:12	979.9851	27.611694
name=2018-06-01-13:13	2018-06-01-13:13	979.96094	27.49384
name=2018-06-01-13:14	2018-06-01-13:14	979.969	27.457966

Figure 46. Data stored in Google Cloud Datastore

Figure 46 shows the data stored in Google Cloud Datastore. Messages stored there as entities, the kind of the data is “messages”, each entity’s name is the date, and the entity includes date, temperature value, and pressure value. For example, as shown in figure, there is an entity shows information include date (2018-06-01-13:05), pressure value (979.9617), and temperature value (27.898651).

	IoT service based on Cloud_A	**IoT service based on Cloud_B**	**IoT service based on Cloud_C**
Communication Method	MQTT	MQTT	MQTT
Received Messages (1 minute)	15	31	50
Publish Speed	0.25/s	0.55/s	0.81/s

Table 9. Comparison result of IoT service based on Clouds

Table 9 shows the comparison result of IoT service based on Clouds in our experiments. For our last experiment (during 1 minute), there are 15 messages published to the topic of Cloud_A, 31 messages published to the topic of Cloud_B, and 50 messages published to the topic of Cloud_C. After many experiments, we considered the message publish speed of Cloud_A is 0.25/s, the message publish speed of Cloud_B is 0.55/s, the message publish speed of Cloud_C is 0.81/s.

Figure 47 shows the tested mobile client of Google Cloud, which is able to pull messages from subscriptions. After click the “Get Message” button, the client is able to pull message from a Google Cloud subscription. For example, as shown in the figure, the client pulled a message include the date (2018-06-04-12:23), pressure value (976.4971923928125), and temperature value (24.289932250976562).

Figure 48 shows the tested mobile client of Google Cloud, which is able to get message entities from Google Cloud Datastore. After click the “Get Message” button, the client is able get an entity from Google Cloud Datastore. For example, as shown in the figure, the client pulled a message include the date (2018-06-01-13:05), pressure value (979.9617), and temperature value (27.898651).

Figure 47. Google Cloud Mobile Client connect with Google Cloud IoT Core

Figure 48. Google Cloud Mobile Client connect with Google Cloud Datastore

Figure 49. The mobile client of Microsoft Azure

Figure 50. The mobile client of AWS

Figure 49 shows the tested mobile client of Microsoft Azure, after click the “Get Message” button, the client is able to get message from the storage account of Azure. For Example, as shown in the figure, the client got a message include the date (2018-06-01-13:50). temperature value (27.673187), and pressure value (979.8434).

Figure 50 shows the tested mobile client of AWS, after click the “Get Message” button, the client is able to get message from the NoSQL Database of AWS Mobile Hub. For Example, as shown in the figure, the client got a message include the date (2018-05-31-07:17), pressure value (975.3861), and temperature value (23.777096).

6. CoT Architecture for Global Cloud and IoT Networks Based on Proxy Using OCF IoTivity and MQTT

6.1 CoT Design for Interworking between Global Cloud and IoT Networks Based on Proxy

Recently, we is really necessary to study the interworking between Cloud and IoT Devices. Figure 51 shows the conceptual model of the architecture for Global Cloud and IoT Networks based on Proxy. The proxy is able to get sensing data from IoT Networks based on IoTivity, and publish or store sensing data message to cloud using MQTT or HTTP protocol. Then Client is able to get the sensing data message from cloud using MQTT or HTTP protocol.

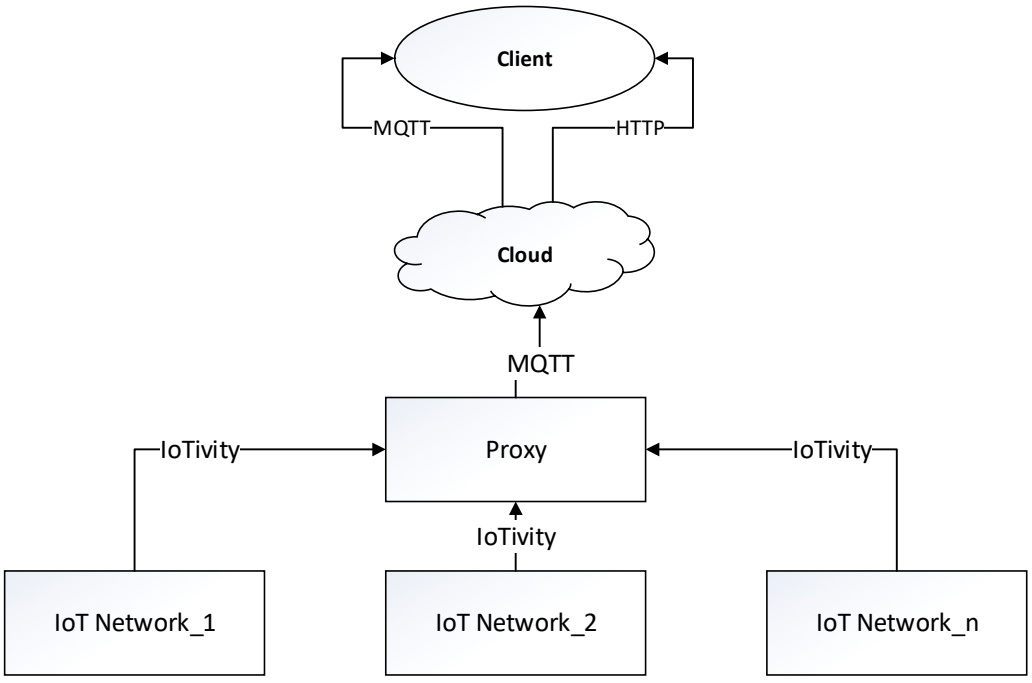


Figure 51. Conceptual model of the architecture for Global Cloud and IoT Networks based on Proxy

Figure 52 shows the configuration of IoT Devices, Sensor Connector is able to get sensing data from sensors via the sensor drivers in Libraries. OCF IoTivity Server is able to make sensing data resource and response sensing data when IoTivity Client requests.

Figure 53 shows the configuration of Proxy. OCF IoTivity Client is able to send request to IoTivity Server, when IoTivity Server response sensing data, Data Buffering will get the sensing data and put the data to Message Conversion. Message Conversion is able to convert the sensing data to message and push the message to MQTT Client. MQTT Client is able to publish this message to the topic in cloud.

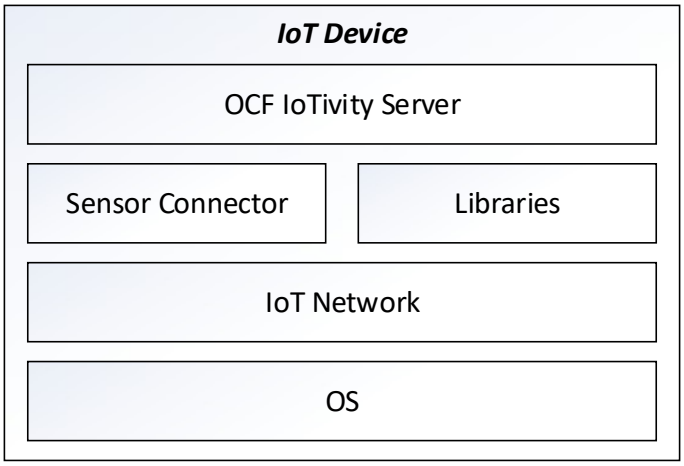


Figure 52. IoT Devices configuration

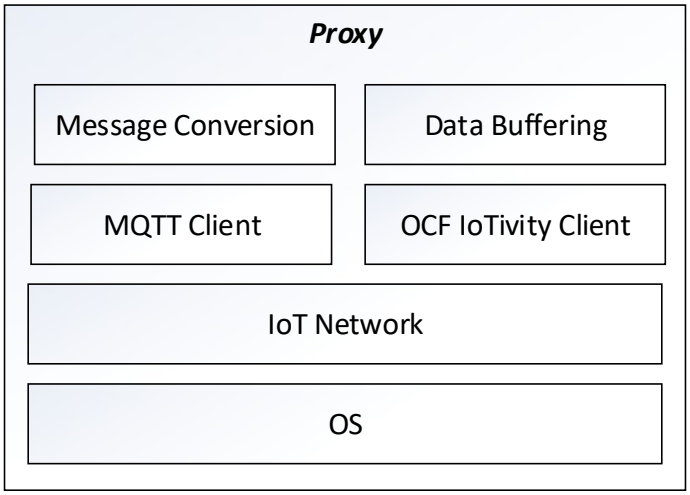


Figure 53. Proxy configuration

Figure 54 shows the sequence diagram of the architecture for Global Cloud and IoT Networks based on Proxy. When the proxy is running, IoTivity Client in Proxy will send request to IoTivity Server in IoT Device, and IoTivity Server will get sensing data from sensors and return the sensing data to IoTivity Client, IoTivity Client will put the sensing data to MQTT Client. MQTT Client will publish the sensing data to the topic in cloud as a message. At last, when the client pull message from a subscription of cloud, the cloud will return message to the client.

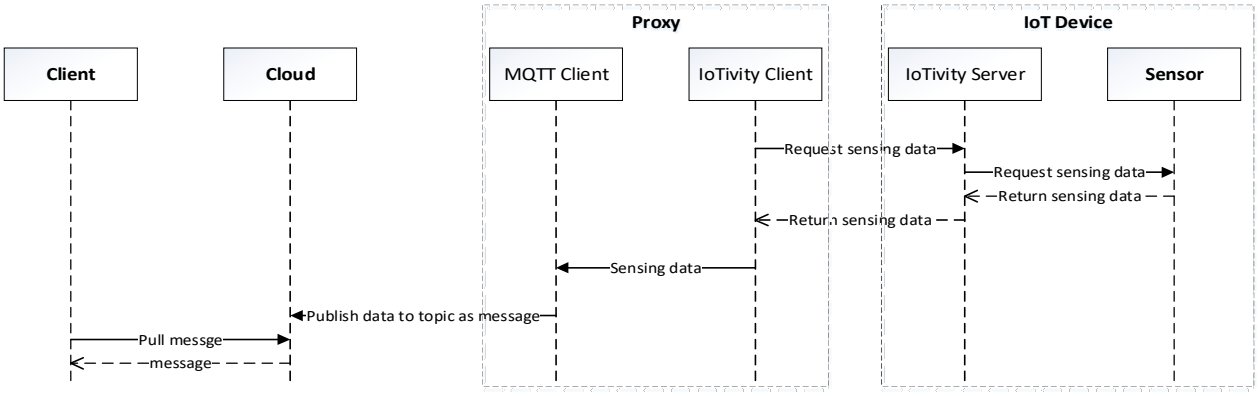


Figure 54. Sequence diagram of the architecture for Global Cloud and IoT Networks based on Proxy

Figure 55 shows the configuration architecture for the connection between IoT network and AWS based on Proxy. We used one Raspberry Pi 3 Model B with Android Things system as Proxy, several other Raspberry Pi 3 Model B boards with Android Things system as IoT devices, and we also used BMP280 Sensor (Temperature, Pressure) to get sensing data. IoT devices get sensing data from BMP280 sensors and respond sensing data when the proxy request. Proxy is able to get sensing data from IoT devices based on IoTivity, and publish sensing data messages to the topic in AWS IoT using MQTT protocol. The MQTT messages in AWS cannot store automatically, so before the device publish message to the topic in AWS, we will create the message item in NoSQL Database of AWS Mobile Hub to store the sensing messages. And then Client is able to get message item from NoSQL Database of AWS Mobile Hub.

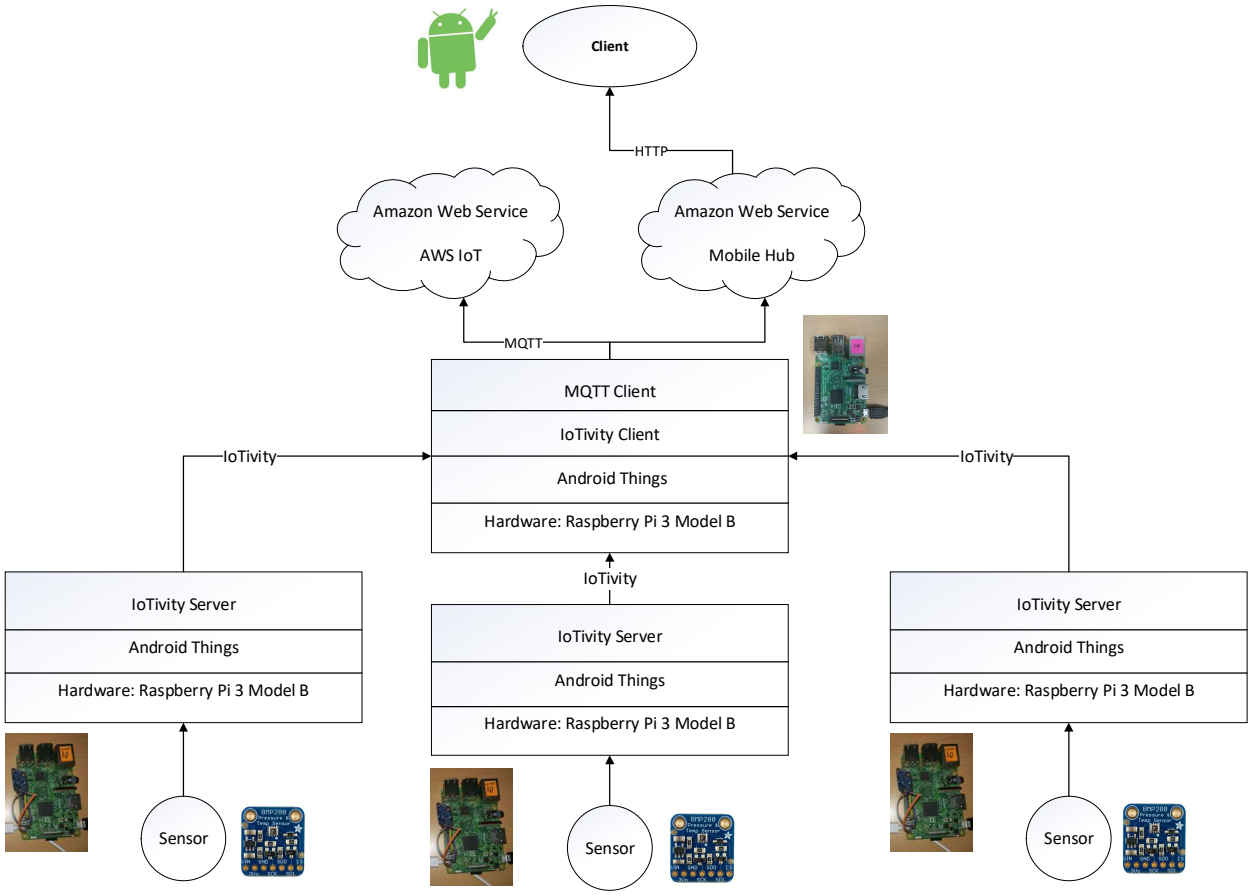


Figure 55. The configuration for the connection between IoT network and AWS based on Proxy

Figure 56 shows the configuration architecture for the connection between IoT network and Azure IoT Hub based on Proxy. We used one Raspberry Pi 3 Model B with Android Things system as Proxy, several other Raspberry Pi 3 Model B boards with Android Things system as IoT devices, and we also used BMP280 Sensor (Temperature, Pressure) to get sensing data. IoT devices get sensing data from BMP280 sensors and respond sensing data when the proxy request. Proxy is able to get sensing data from IoT devices based on IoTivity, and publish sensing data messages to the topic in Azure IoT Hub using MQTT protocol. After the messages published to the topic of Azure, the messages will be stored in the Blobs of Storage Account of Azure, and Client is able to get message files from Blobs of Azure.

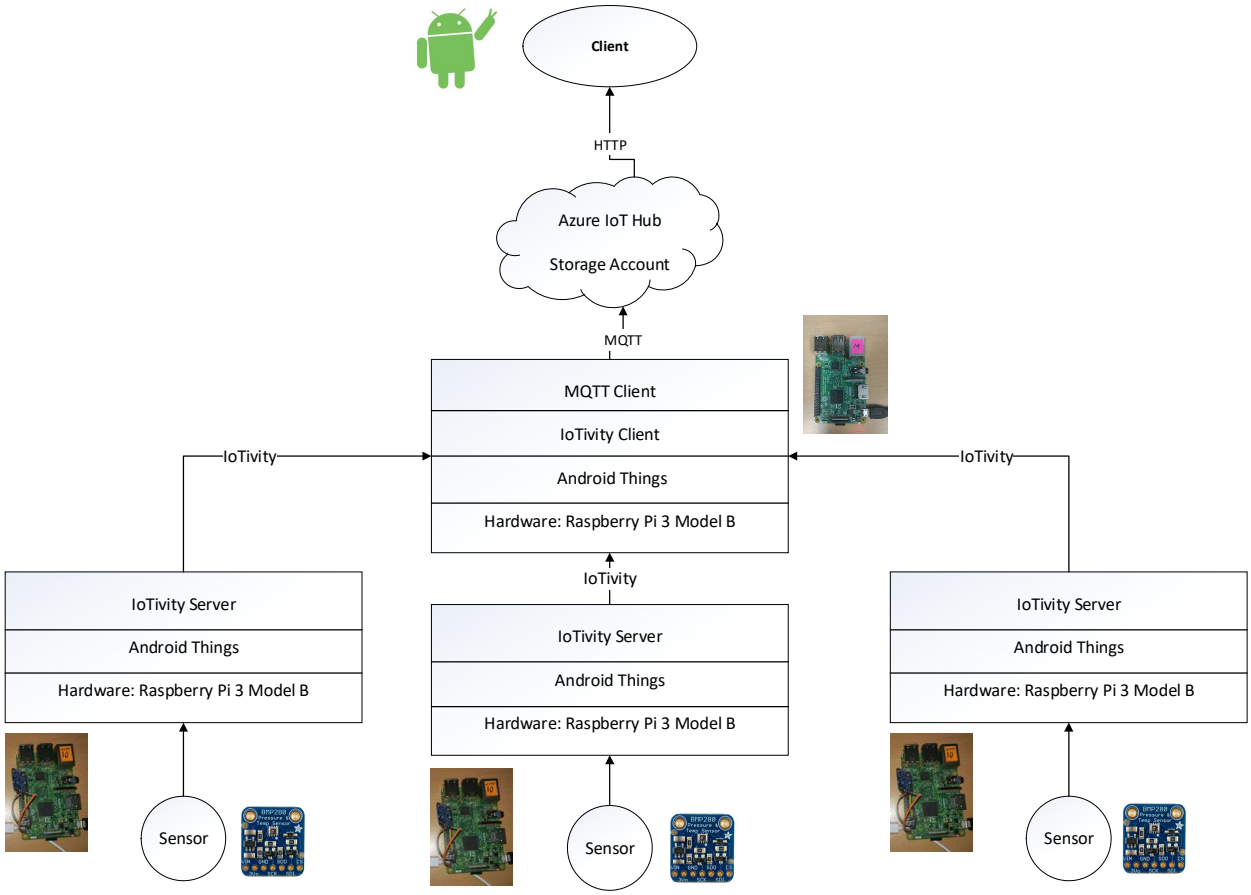


Figure 56. The configuration for the connection between IoT network and Azure IoT Hub based on Proxy

Figure 57 shows the configuration architecture for the connection between IoT network and Google Cloud based on Proxy. We used one Raspberry Pi 3 Model B with Android Things system as Proxy, several other Raspberry Pi 3 Model B boards with Android Things system as IoT devices, and we also used BMP280 Sensor (Temperature, Pressure) to get sensing data. IoT devices get sensing data from BMP280 sensors and respond sensing data when the proxy request. Proxy is able to get sensing data from IoT devices based on IoTivity, and publish sensing data messages to the topic in Google Cloud IoT Core using MQTT protocol. The messages published to topic in Google Cloud will disappear after several days, so if we want to store the messages, we need to use Google Cloud Datastore. When we publish the message to the topic in Google Cloud, we also create a message entity in Google Cloud Datastore. Client is able to pull message

from subscriptions in Google Cloud, if the message has been disappeared, Client is able to get message entity from Google Cloud Datastore.

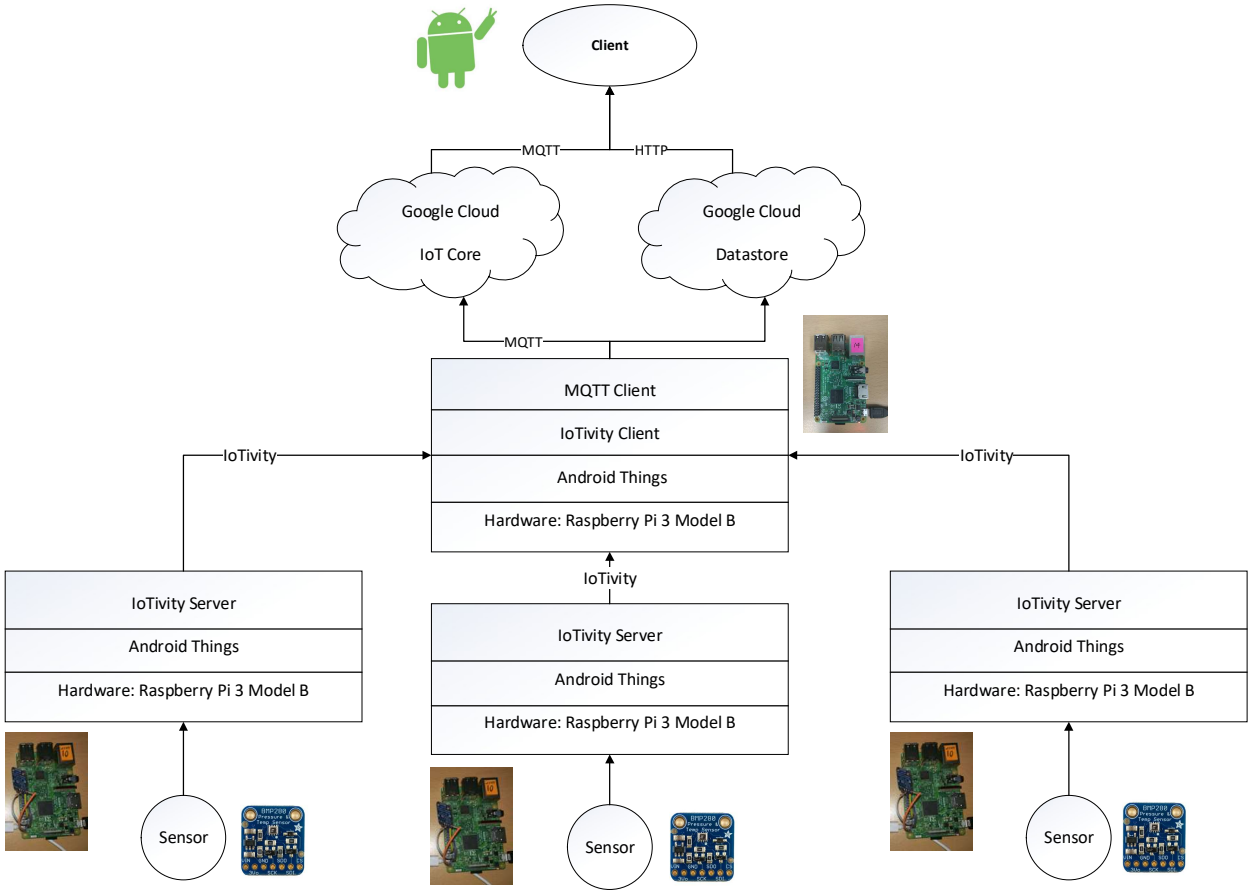


Figure 57. The configuration for the connection between IoT network and Google Cloud based on Proxy

6.2 Experiments and Results of CoT Architecture for Global Cloud and IoT Networks Based on Proxy

Table 10 shows the configuration environment of IoT Device. The Android Things system we used is version 0.6.1, and we also used Android SDK 26, Java 1.8, IoTivity 1.2.1, Google Bmx280 Sensor Driver 0.4.

Components	Version
Android Things	0.6.1
Android SDK	26
Java	1.8
IoTivity	1.2.1
Bmx280 Sensor Driver	0.4

Table 10. Configuration environment of IoT Device

Table 11 shows the configuration environment of Proxy. Compare with the other boards, we add cloud SDKs instead of the Bmx 280 sensor driver, include Microsoft Azure IoT SDK 1.6.0, AWS Android SDK 2.6.11, Google Cloud Platform SDK 1.22.0.

Components	Version
Android Things	0.6.1
Android SDK	26
Java	1.8
IoTivity	1.2.1
Microsoft Azure IoT SDK	1.6.0
AWS Android SDK	2.6.11
Google Cloud Platform SDK	1.22.0

Table 11. Configuration environment of Proxy

Table 12 shows the development environment of mobile clients based on Proxy. We used Android Studio 3.0.1, Android SDK 27, AWS Java SDK 1.11.293, Azure IoT SDK 1.7.23, Google Cloud PubSub 0.45.0, Google Cloud Storage 1.26.0, and Google Cloud Datastore 1.31.0.

Components	Version
Android Studio	3.0.1
Android SDK	27
AWS Java SDK	1.11.293
Azure IoT SDK	1.7.23
Google Cloud PubSub	0.45.0
Google Cloud Storage	1.26.0
Google Cloud Datastore	1.31.0

Table 12. Development environment of mobile clients based on Proxy

Figure 58 shows the IoT devices we used in architecture for Global Cloud and IoT Networks based on Proxy. We installed Android Things in Raspberry Pies, and we used BMP280 pressure and temperature sensor. Figure (a) shows the proxy board, figure (b) shows the normal Android Things boards, and figure (c) shows the BMP280 sensor we used.

(a) Raspberry Pi (Proxy)

(b) Raspberry Pi (IoT Devices)

(c) BMP 280 Sensor

Figure 58. IoT devices used in architecture for Global Cloud and IoT Networks based on Proxy

Figure 59 shows the connection information of AWS in code of Proxy. The information includes customer specific endpoint, Cognito pool ID, AWS IoT policy name, region, key store name, key store password and certificate ID.

```
private static final String CUSTOMER_SPECIFIC_ENDPOINT = "a2nnptnxwb37rq.iot.us-east-2.amazonaws.com";
// Cognito pool ID. For this app, pool needs to be unauthenticated pool with
// AWS IoT permissions.
private static final String COGNITO_POOL_ID = "us-east-2:4f5627c9-e1be-4563-8606-9ac2460ffc90";
// Name of the AWS IoT policy to attach to a newly created certificate
private static final String AWS_IOT_POLICY_NAME = "AndroidThing";

// Region of AWS IoT
private static final Regions MY_REGION = Regions.US_EAST_2;
// Filename of KeyStore file on the filesystem
private static final String KEYSTORE_NAME = "iot_keystore";
// Password for the private key in the KeyStore
private static final String KEYSTORE_PASSWORD = "password";
// Certificate and key aliases in the KeyStore
private static final String CERTIFICATE_ID = "default";
```

Figure 59. Connection information of AWS based on Proxy

Figure 60 shows the result (sensing data) of cloud (AWS). the message has published successfully. And as shown in the figure, the message include date (2018-02-09-11:24:35), temperature value (24.37198), and pressure value (807.8578).

Figure 60. Results of cloud (AWS) based on Proxy

Figure 61 shows the successful connections in 1 hour with AWS based on Proxy, When the connections become stable, the message publish speed is 0.1/s, the minimum speed is 0/s, the maximum speed is 0.1/s.

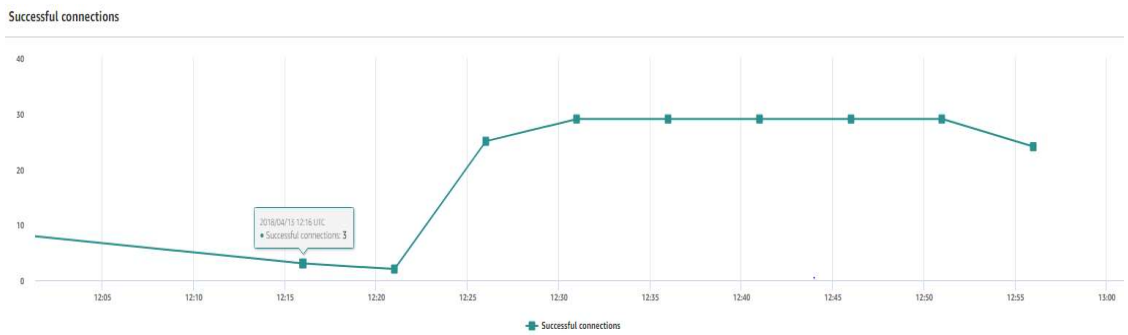


Figure 61. The successful connections in 1 hour with AWS based on Proxy

Figure 62 shows the data stored in AWS NoSQL database based on Proxy. The name of the table is finaltest-mobilehub-887317787-messages, the primary sort key is the date, each item includes date, temperature value, and pressure value. For example, as shown in figure, there is an item shows information include date (2018-06-01-13:53), pressure value (979.73376), and temperature value (28.216324).

Scan: [Table] finaltest-mobilehub-887317787-messages: ... Viewing 101 to 187 items

Scan [Table] finaltest-mobilehub-887317787-messages: userId, date

Add filter

Start search

userId	date	pressure	temperature
raspberry-1	2018-05-31-12:58	807.5527	24.115576
raspberry-1	2018-05-31-12:59	807.5709	24.136087
raspberry-1	2018-05-31-13:00	807.38806	23.987368
raspberry-1	2018-05-31-13:01	807.38806	23.992496
raspberry-1	2018-06-01-13:48	979.7905	27.908895
raspberry-1	2018-06-01-13:49	979.7643	27.796167
raspberry-1	2018-06-01-13:50	979.8434	27.673187
raspberry-1	2018-06-01-13:51	979.7241	28.00625
raspberry-1	2018-06-01-13:52	979.8033	28.2983
raspberry-1	2018-06-01-13:53	979.73376	28.216324

Figure 62. Data stored in AWS NoSQL database based on Proxy

Figure 63 shows the connection information of Azure in code of Proxy. The information includes connection string (primary key) and device ID.

```
private final String connString = "HostName=my-hub.azure-devices.net;DeviceId=raspberry-pi;SharedAccessKey=OhJ3uXxxK54BDEI9u/jO5iCmG051dDeGBJuOP/r3ZGI=";
private final String deviceId = "raspberry-pi";
```

Figure 63. Connection information of Azure based on Proxy

Figure 64 shows the received messages in Azure based on Proxy, it's not able to show messages directly, we need to download the message file. Figure (a) is the Blob service of the storage we created, it is able to find the messages folder, and figure (b) shows when we click the messages file, it's able to click the "Download" button to download the messages file.

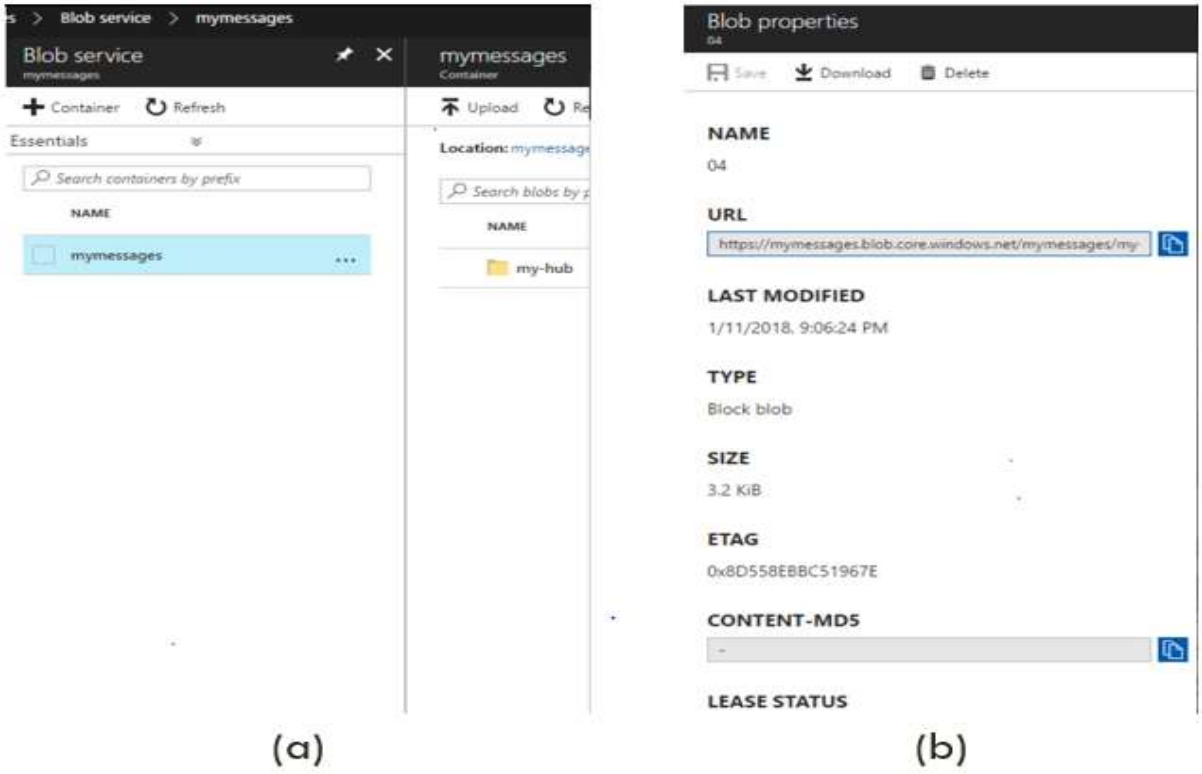


Figure 64. Received message file in Azure based on Proxy

Figure 65 shows the downloaded sensing data file based on Proxy, the message has published successfully. And as shown in the figure, the message include date (2018-02-09-11:59:41), temperature value (24.16173), and pressure value (807.6015).

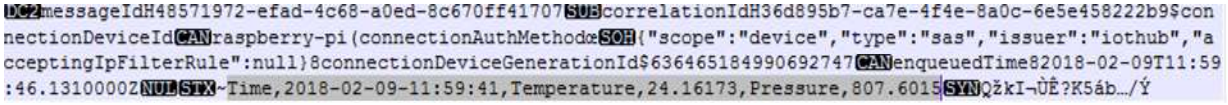


Figure 65. Result file of Azure based on Proxy

Figure 66 shows the sum messages delivered to storage endpoints of Azure based on Proxy. When the connections become stable, the message publish speed is 0.04/s, the minimum speed is 0/s, the maximum speed is 0.04/s.

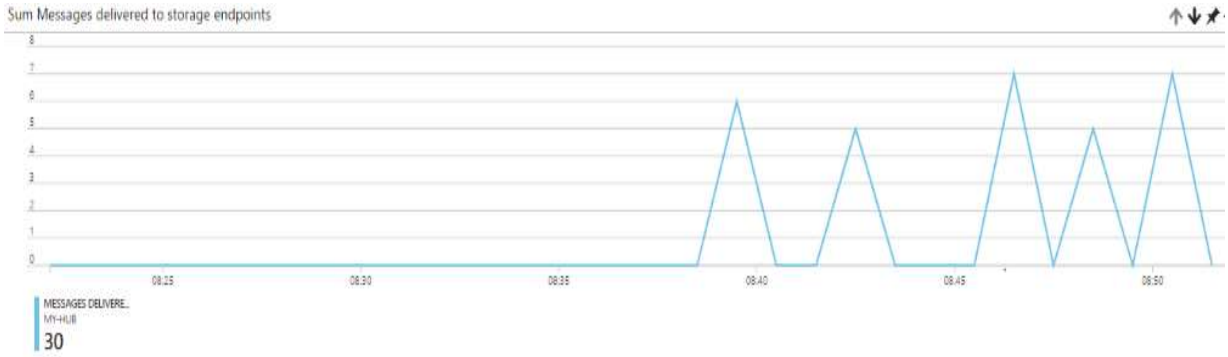


Figure 66. Sum messages delivered to storage endpoints of Azure based on Proxy

Figure 67 shows the connection information of Google Cloud Platform in code of Proxy. The information includes default bridge hostname, default bridge port, project ID, Registry ID, Device ID, region, topic format, client ID format and broker URL format.

```
private static final String DEFAULT_BRIDGE_HOSTNAME = "mqtt.googleapis.com";
private static final short DEFAULT_BRIDGE_PORT = 8883;

private static final String PROJECT_ID = "raspberrypi-187702";
private static final String REGISRY_ID = "my-registry-03";
private static final String DEVICE_ID = "my-raspberry01";
private static final String REGION = "asia-east1";

public static final String UNUSED_ACCOUNT_NAME = "unused";
private static final String MQTT_TOPIC_FORMAT = "/devices/%s/events";
private static final String MQTT_CLIENT_ID_FORMAT = "projects/%s/locations/%s/registries/%s/devices/%s";
private static final String BROKER_URL_FORMAT = "ssl://%s:%d";
```

Figure 67. Connection information of Google Cloud Platform based on Proxy

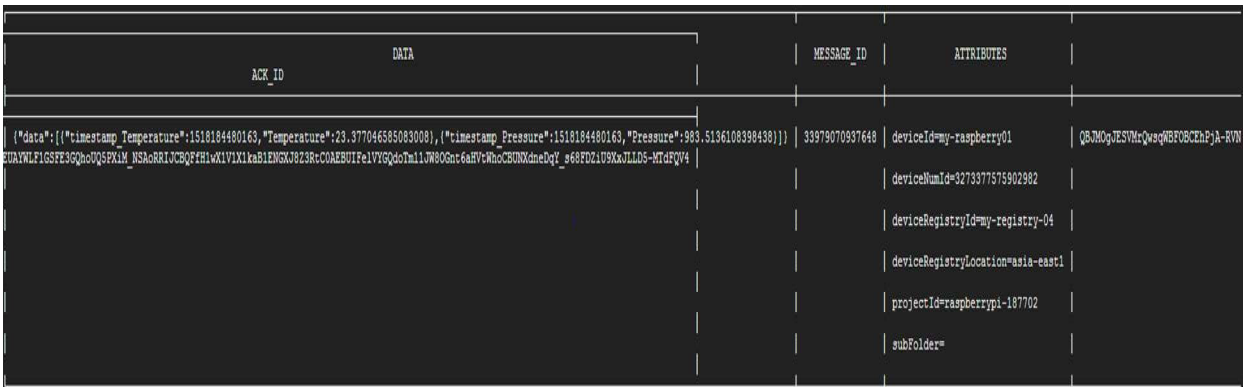


Figure 68. Published message in topic of Google Cloud based on Proxy

Figure 68 shows the result in Google Cloud Platform based on Proxy, the message has published successfully. We need shell command to pull the message from a subscription, and as shown in figure, there is a message include device ID, device registry ID, device registry location, project ID, temperature value, and pressure value.

Figure 69 shows the publish request count of Pub/Sub topic in Google Cloud Platform based on Proxy. When the connections become stable, the message publish speed is 0.033/s, the minimum speed is 0/s, the maximum speed is 0.1/s.

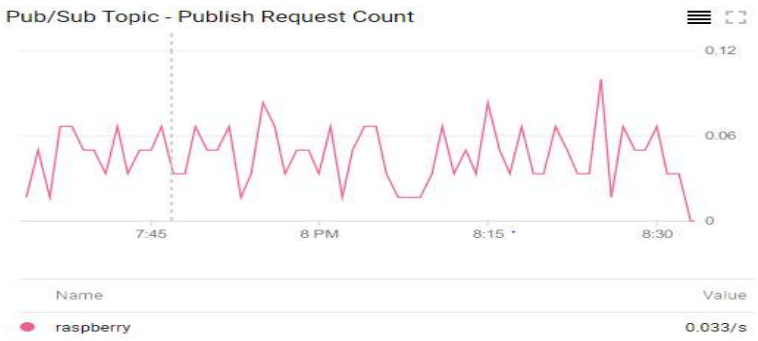


Figure 69. The publish request count of Pub/Sub topic in Google Cloud Platform based on Proxy

Entities CREATE ENTITY REFRESH DELETE

name=2018-06-04-12:03	2018-06-04-12:03	976.3289	27.140238
name=2018-06-04-12:04	2018-06-04-12:04	976.41974	27.283728
name=2018-06-04-12:05	2018-06-04-12:05	811.6852	27.580952
name=2018-06-04-12:06	2018-06-04-12:06	811.69745	27.591198
name=2018-06-04-12:07	2018-06-04-12:07	811.42267	27.375973
name=2018-06-04-12:08	2018-06-04-12:08	810.3298	26.453457
name=2018-06-04-12:09	2018-06-04-12:09	809.80475	26.022884
name=2018-06-04-12:10	2018-06-04-12:10	809.05396	25.387207
name=2018-06-04-12:11	2018-06-04-12:11	808.96234	25.310303
name=2018-06-04-12:12	2018-06-04-12:12	976.61	25.002678

Figure 70. Data stored in Google cloud Datastore based on Proxy

Figure 70 shows the data stored in Google cloud Datastore based on Proxy. Messages stored there as entities, the kind of the data is “messages”, each entity’s name is the date, and the entity includes date, temperature value, and pressure value. For example, as shown in figure, there is an entity shows information include date (2018-06-04-12:03), pressure value (976.3289), and temperature value (27.140238).

Figure 71. The mobile client of Google Cloud for Google Cloud IoT Core based on Proxy

Figure 71 shows the tested mobile client of Google Cloud for Google Cloud IoT Core based on Proxy, which is able to pull messages from subscriptions. After click the "Get Message" button, the client is able to pull message from a Google Cloud subscription. For example, as shown in the figure, the client pulled a message include the date (2018-05-27-06:23), pressure value (953.4248820797152), and temperature value (27.241511379901467).

Figure 72 shows the tested mobile client of Google Cloud for Google Cloud Datastore based on Proxy, which is able to get message entities from Google Cloud Datastore. After click the "Get Message" button, the client is able get an entity from Google Cloud Datastore. For example, as shown in the figure, the client pulled a message include the date (2018-06-04-12:03), pressure value (976.3289), and temperature value (27.140238).

Figure 72. The mobile client of Google Cloud for Google Cloud Datastore based on Proxy

Figure 73 shows the tested mobile client of Microsoft Azure based on Proxy, after click the "Get Message" button, the client is able to get message from the storage account of Azure. For Example, as shown in the figure, the client got a message include the date (2018-06-04-12:07). temperature value (27.375973), and pressure value (811.42267).

Figure 74 shows the tested mobile client of AWS based on Proxy, after click the "Get Message" button, the client is able to get message from the NoSQL Database of AWS Mobile Hub. For Example, as shown in the figure, the client got a message include the date (2018-16-01-13:48), pressure value (979.7905), and temperature value (27.908895).

Figure 73. The mobile client of Microsoft Azure based on Proxy

Figure 74. The mobile client of AWS based on Proxy

7. CoT Architecture for Vehicle Monitoring and Control Service Based on Proxy Using OCF IoTivity and MQTT

7.1 CoT Design for Vehicle Monitoring and Control Service Based on Proxy

Electric vehicles have been used more and more wildly. In this paper, we present the mobile control service of electric vehicles based on CoV architecture, which is able to provide users monitor service and control service, and upload vehicle states to cloud.

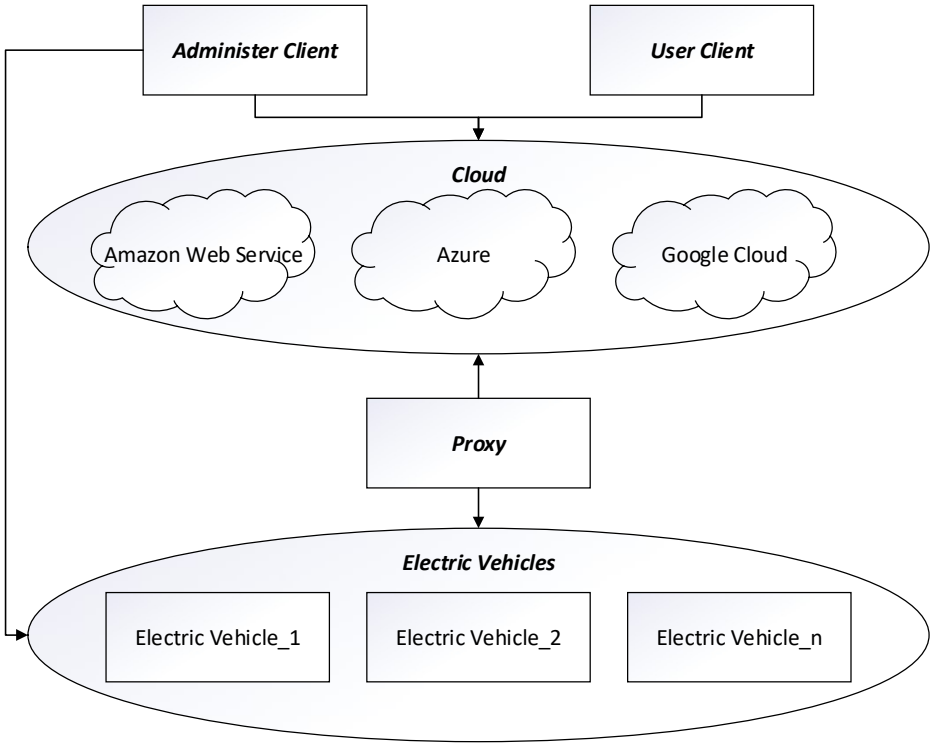


Figure 75. Conceptual model of CoT architecture for vehicle monitoring and control service

Figure 75 shows the conceptual model of CoT architecture for vehicle monitoring and control service. The proxy is able to get vehicle state from electric vehicles and upload the state information to cloud (AWS, Azure, Google Cloud), the clients are able to get vehicle state information from cloud and show the state to users. The vehicle state information include battery state (present), door state (lock or unlock), trunk state (lock or unlock), audio state (turn on or turn off), head lights state (turn on or turn off), interior light state ((turn on or turn off)), alarm state ((turn on or turn off)), and the engine state (start or stop). The administer client is able to control electric vehicles, include lock or unlock the doors, lock or unlock the trunk, turn on or turn off the audio, turn on or turn off the head lights, turn on or turn off the interior light, turn on or turn of the alarm, and start or stop the engine.

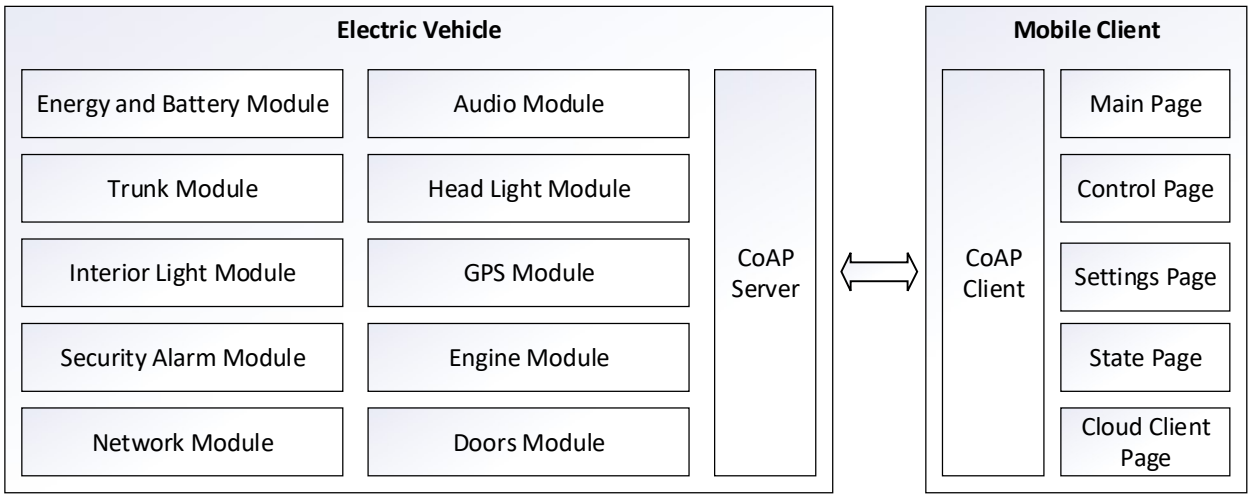


Figure 76. The design of the vehicle CoT architecture

Figure 76 shows the design of the vehicle CoT architecture. The system consists of two parts: Vehicle Emulator and App Client. In Vehicle Emulator, there are ten modules include energy and Battery Module, Trunk Module, Interior Light Module, Security Module, Network Module, Audio Module, Head Light Module, GPS Module, Engine Module, and Doors Module, there is also a CoAP Server in vehicle Emulator. Energy and Battery Module is able to check the state of the battery. Audio Module is able to turn on/off the audio. Trunk Module is able to lock/unlock the trunk. Head Light Module is able to turn on/off the head light. Interior Light Module is able to turn on/off the interior light. GPS Module is able to turn on/off the

GPS. Security Alarm Module is able to start/stop the security service (security alarm). Engine Module is able to start/stop the engine. Network Module is able to set up the network settings and turn on/off network connection. Doors Module is able to lock/unlock the doors. CoAP Server is able to get the request from Client and respond accordingly. In App Client, there are four modules include Main Page Module, Control Page Module, Settings Page Module, and State Page Module, there is also a CoAP Client in App Client. The main page is able to show all functions of this app (Control, Settings, State). The control page is able to control the Vehicle Emulator. The settings page is able to let users set the IP address of the Vehicle Emulator. The state page is able to show the battery state of the Vehicle Emulator. CoAP Client is able to send request to control Vehicle Emulator or get battery state from Vehicle Emulator.

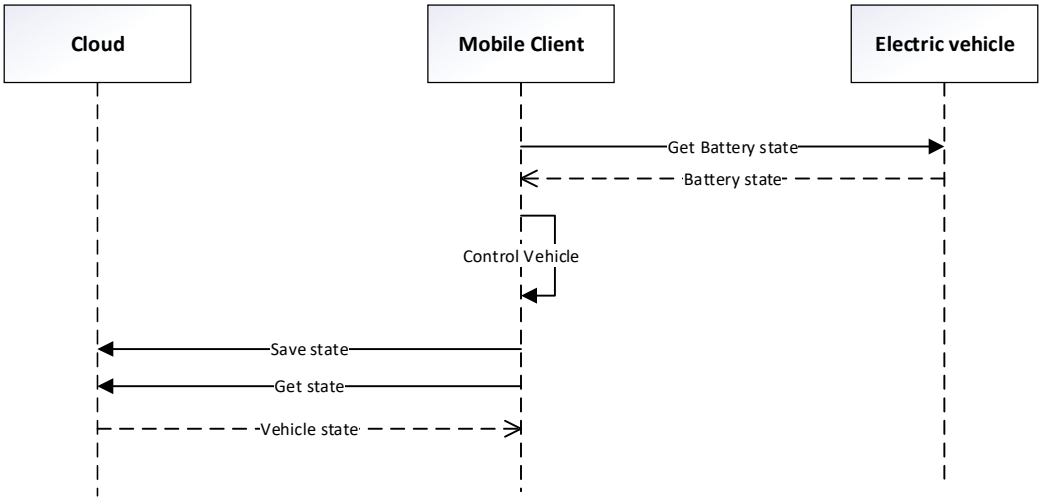


Figure 77. The sequence diagram for monitoring electric vehicle

Figure 77 shows the sequence diagram for monitoring electric vehicle. When Mobile Client send “get battery state” request to Vehicle Emulator, Vehicle Emulator will return battery state to App Client. After the client controls the vehicles, it is able to upload the vehicle state to cloud. When the client request vehicle state from cloud, the cloud is able to return the state of vehicles.

Figure 78 shows the sequence diagram for controlling electric vehicle. When App Client send “start/stop engine” request to Vehicle Emulator, Vehicle Emulator will start/stop engine and return true/false to App

Client. When App Client send “turn on/off audio” request to Vehicle Emulator, Vehicle Emulator will turn on/off the audio and return true/false to App Client. When App Client send “lock/unlock doors” request to Vehicle Emulator, Vehicle Emulator will lock/unlock the doors and return true/false to App Client. When App Client send “lock/unlock trunk” request to Vehicle Emulator, Vehicle Emulator will lock/unlock the trunk and return true/false to App Client. When App Client send “turn on/off head light” request to Vehicle Emulator, Vehicle Emulator will turn on/off the head light and return true/false to App Client. When App Client send “turn on/off interior light” request to Vehicle Emulator, Vehicle Emulator will turn on/off the interior light and return true/false to App Client. When App Client send “turn on/off alarm” request to Vehicle Emulator, Vehicle Emulator will turn on/off the alarm and return true/false to App Client. When App Client send “turn on/off GPS” request to Vehicle Emulator, Vehicle Emulator will turn on/off GPS and return true/false to App Client.

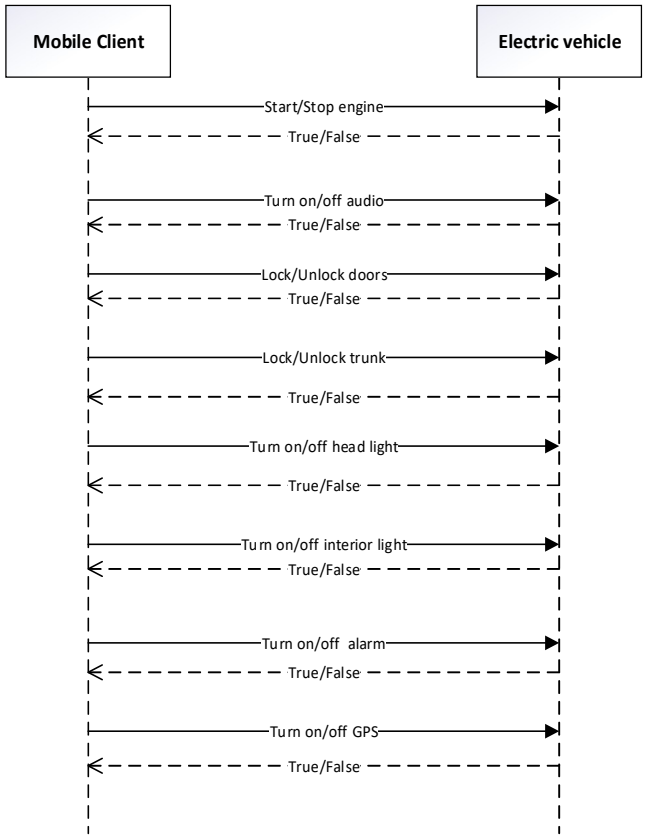


Figure 78. The sequence diagram for controlling electric vehicle

7.2 Experiments and Results of CoT Architecture for Vehicle Monitoring and Control Service Based on Proxy

Table 13 shows the development environment of vehicle emulator.

Component	Version
Windows OS	10
Visual Studio	2015
CoAP.NET	v4.0.30319

Table 13. Development environment of IoT Vehicle System

Table 14 shows the development environment of mobile client.

Component	Version
Android Studio	3.0.1
Android SDK	27
org.eclipse.californium	2.0
AWS Android SDK	2.6
Azure IoT SDK	1.7.23
Google Cloud Datastore	1.31.0

Table 14. Development environment of mobile client

Figure 79 shows the implement design of CoT architecture for vehicle monitoring and control service. There is one app that include the user client part, administer client part, and proxy part. This app is able to monitor the vehicle emulator, control the vehicle emulator, upload the state of vehicle emulator to cloud, and get the state of vehicle emulator from cloud.

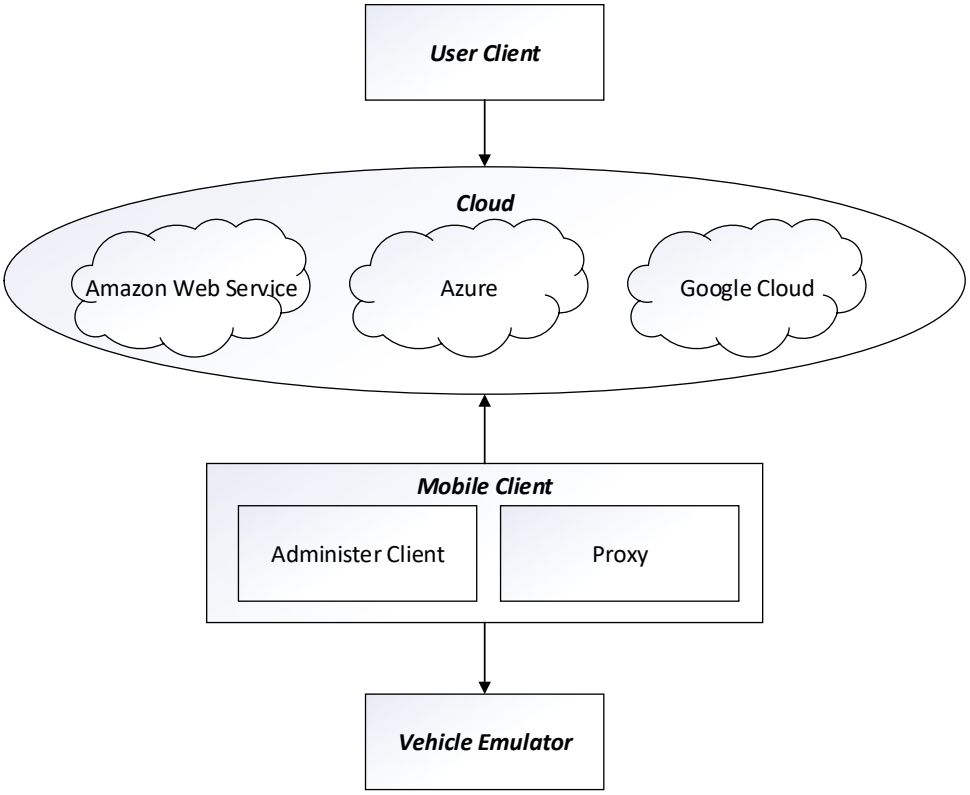


Figure 79. Implement design of CoT architecture for vehicle monitoring and control service

Figure 80 shows the implementation result of Vehicle Emulator. Figure (a) shows the initial state, figure (b) shows the state that all modules have been opened.

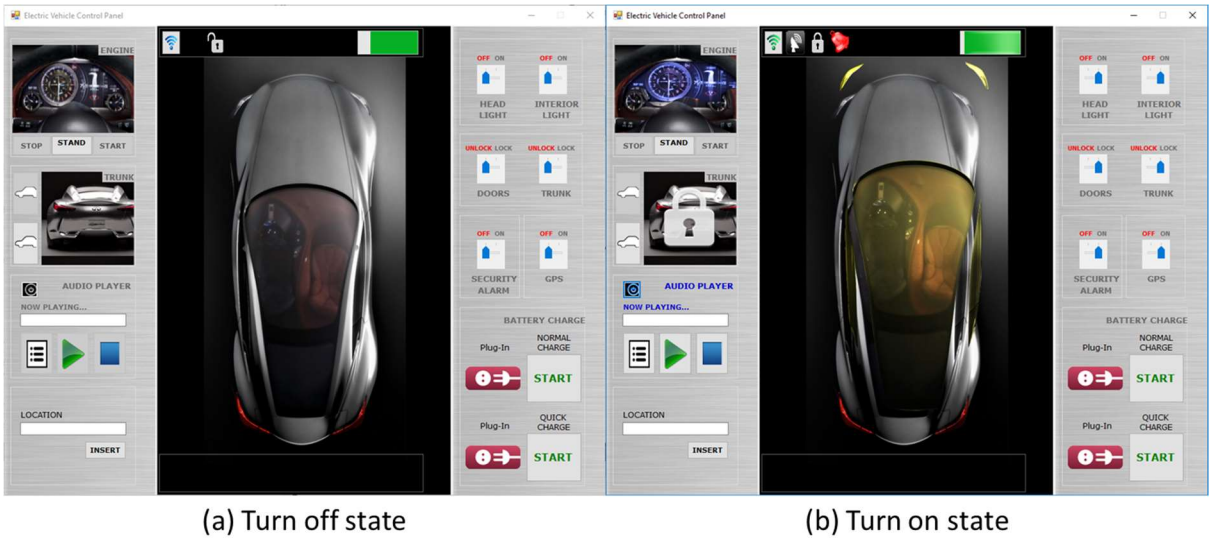


Figure 80. Implementation results of vehicle Emulator

Figure 81 shows the implementation result for monitoring part of mobile client. Figure (a) shows the main page, there are four buttons in the main page, which is able to visit the control page, the monitor mage, the setting page, and the cloud client page. Figure (b) shows the state page, which is able to show the battery state of Vehicle Emulator.

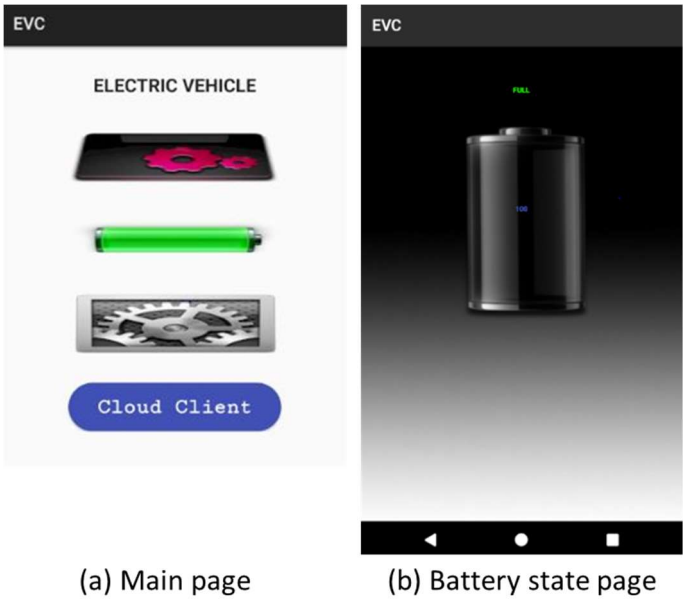


Figure 81. Implementation results for monitoring part of mobile client

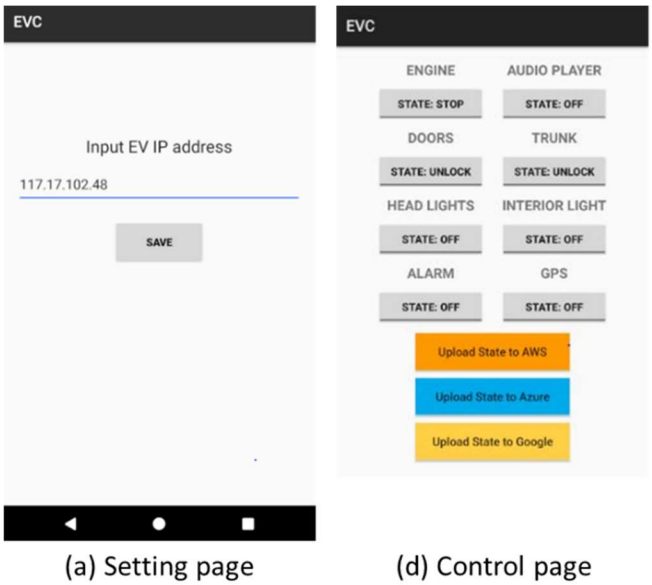
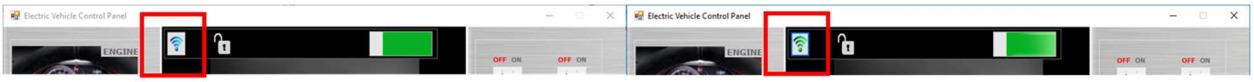


Figure 82. Implementation results for controlling part of mobile client

Figure 82 shows the implementation result for controlling part of mobile client. Figure (a) shows the setting page, which is able to let users input the IP address of Vehicle Emulator. Figure (b) shows the control page, there are eight buttons to control different modules include engine, audio player, doors, trunk, head lights, interior light, alarm, and GPS. Under the control buttons, there are three buttons which are able to upload current vehicle state to cloud (AWS, Azure, Google Cloud Platform).

Figure 83 shows how to start the network (CoAP) connection. Click the image in figure (a) is able to start the connection, and the figure will change as figure (b) (the color of the internet icon will change from blue to green).



(a) Stop Internet connection (b) Start Internet connection

Figure 83. The way to start the network connection

Figure 84 shows how to control the engine. Click the stand button and then click the start button will start the engine, click the stop button will stop the engine. Figure (a) shows the engine-start state, figure (b) shows the engine-stop state.

Figure 84. The way to control the engine

Figure 85 shows how to control the audio player. Click the small icon will start the player, click the icon again will stop the player. Figure (a) shows the player-on state, figure (b) shows the player-off state.

(a) Audio on (b) Audio off

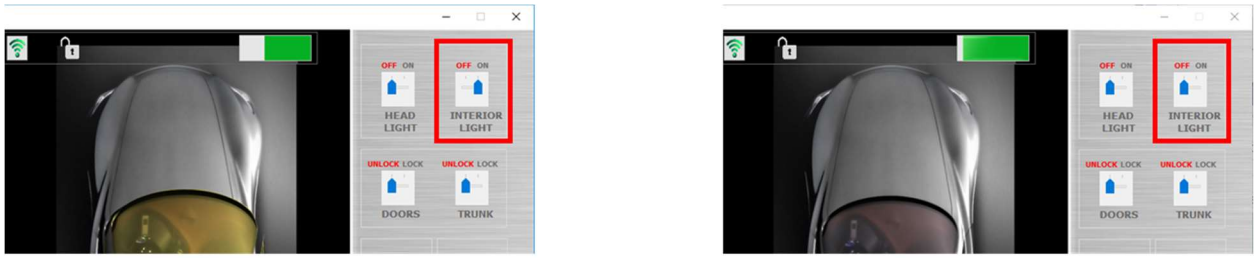
Figure 85. The way to control the audio player

Figure 86 shows how to control the head lights. Move the marker to “ON” will turn on the head lights, move the marker to “OFF” will turn off the head lights. Figure (a) shows the lights-on state, figure (b) shows the lights-off state.

(a) Head lights on (b) Head lights off

Figure 86. The way to control the head lights

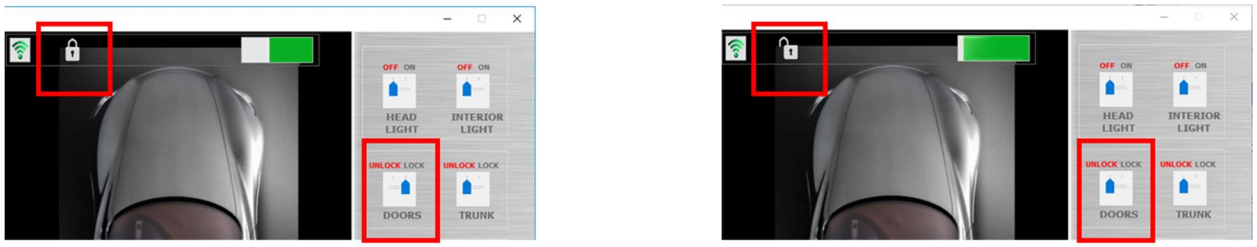
Figure 87 shows how to control the interior light. Move the marker to “ON” will turn on the interior light, move the marker to “OFF” will turn off the interior light. Figure (a) shows the light-on state, figure (b) shows the light-off state.



(a) Interior lights on (b) Interior lights off

Figure 87. The way to control the interior light

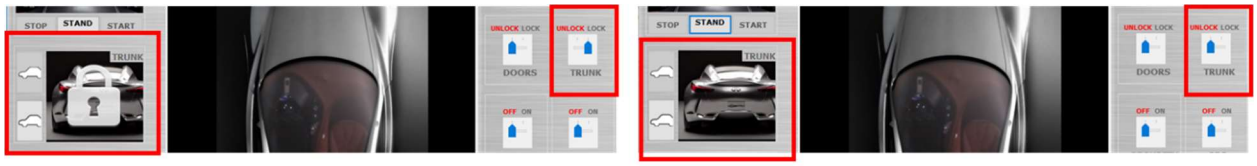
Figure 88 shows how to control the doors. Move the marker to “LOCK” will lock the doors and the lock icon will turn to locked, move the marker to “UNLOCK” will unlock the doors and the lock icon will turn to unlocked. Figure (a) shows the doors-lock state, figure (b) shows the doors-unlock state.



(a) Doors locked (b) Doors unlocked

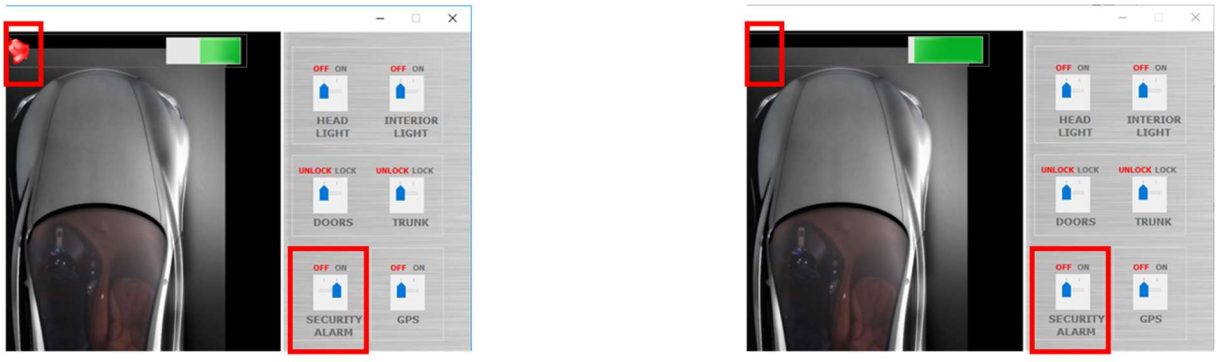
Figure 88. The way to control the doors

Figure 89 shows how to control the trunk. Move the marker to “LOCK” will lock the trunk and the trunk icon will turn to locked, move the marker to “UNLOCK” will unlock the trunk and the trunk icon will turn to unlocked. Figure (a) shows the trunk -lock state, figure (b) shows the trunk -unlock state.



(a) Trunk locked (b) Trunk unlocked

Figure 89. The way to control the trunk

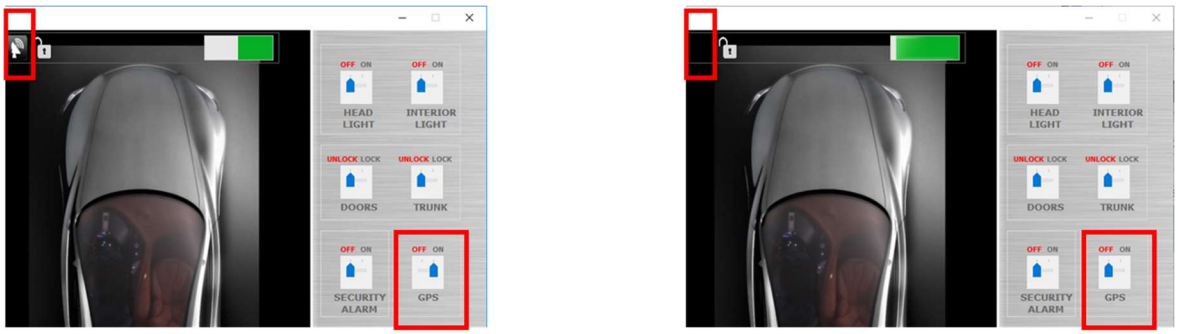


(a) Alarm on (b) Alarm off

Figure 90. The way to control the security alarm

Figure 90 shows how to control the security alarm. Move the marker to “ON” will turn on the alarm, move the marker to “OFF” will turn off the alarm. Figure (a) shows the alarm-on state, figure (b) shows the alarm-off state.

Figure 91 shows how to control the GPS. Move the marker to “ON” will turn on the GPS, move the marker to “OFF” will turn off the GPS. Figure (a) shows the GPS-on state, figure (b) shows the GPS-off state.



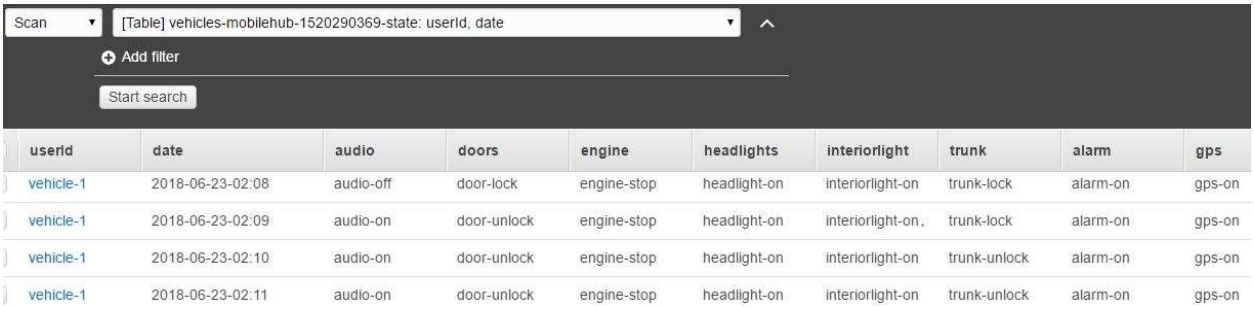
(a) GPS on (b) GPS off

Figure 91. The way to control the GPS

After controlling the vehicles, the clients are able to upload current vehicle state to clouds (AWS, Azure, Google Cloud Platform). Figure 92 shows the stored state in the NoSQL Database of AWS Mobile Hub. For example, on the date “2018-06-23-02:08”, on the vehicle “vehicle-1”, the audio was off, the doors were locked, the engine was stop, the head lights were on, the interior light was on, the trunk was locked, the alarm was on, and the GPS was on.

Figure 93 shows the stored state in Azure Storage Account. For example, on the date “2018-06-23-04:56”, on the vehicle “vehicle-1”, the engine was stop, the audio was off, the door was unlocked, the trunk was unlocked, the head lights were off, the interior light was off, the alarm was off, the GPS was off.

Figure 94 shows the stored state in Google Cloud Datastore. For example, on the date “2018-06-23-04:23”, on the vehicle, the alarm was off, the audio was on, the doors were unlocked, the engine was start, the GPS was off, the headlight was on, the interior light was on, the trunk was unlocked.



userId	date	audio	doors	engine	headlights	interiorlight	trunk	alarm	gps
vehicle-1	2018-06-23-02:08	audio-off	door-lock	engine-stop	headlight-on	interiorlight-on	trunk-lock	alarm-on	gps-on
vehicle-1	2018-06-23-02:09	audio-on	door-unlock	engine-stop	headlight-on	interiorlight-on	trunk-lock	alarm-on	gps-on
vehicle-1	2018-06-23-02:10	audio-on	door-unlock	engine-stop	headlight-on	interiorlight-on	trunk-unlock	alarm-on	gps-on
vehicle-1	2018-06-23-02:11	audio-on	door-unlock	engine-stop	headlight-on	interiorlight-on	trunk-unlock	alarm-on	gps-on

Figure 92. The stored vehicle state in AWS NoSQL Database

Figure 93. The stored vehicle state file in Azure Storage Account

Name/ID	alarm	audio	date	doors	engine	gps	headlights	interiorlight	trunk
name=2018-06-23-04:08	alarm-off	audio-off	2018-06-23-04:08	door-unlock	engine-stop	gps-off	headlight-off	interiorlight-off	trunk-unlock
name=2018-06-23-04:09	alarm-on	audio-on	2018-06-23-04:09	door-lock	engine-start	gps-off	headlight-off	interiorlight-off	trunk-lock
name=2018-06-23-04:10	alarm-on	audio-on	2018-06-23-04:10	door-lock	engine-start	gps-on	headlight-on	interiorlight-on	trunk-lock
name=2018-06-23-04:11	alarm-on	audio-off	2018-06-23-04:11	door-unlock	engine-stop	gps-on	headlight-on	interiorlight-on	trunk-unlock
name=2018-06-23-04:23	alarm-off	audio-on	2018-06-23-04:23	door-unlock	engine-start	gps-off	headlight-on	interiorlight-on	trunk-unlock
name=2018-06-23-04:26	alarm-on	audio-off	2018-06-23-04:26	door-lock	engine-stop	gps-on	headlight-on	interiorlight-on	trunk-lock

Figure 94. The stored vehicle state in Google Cloud Datastore

When users want to get vehicle state from cloud, they are able to enter the Cloud Client page from the main page. There are three buttons, which are able to get vehicle state from AWS, Azure, and Google Cloud Platform. Figure 95 shows the vehicle state got from AWS, on the date “2018-06-23-02:10”, on the vehicle,

the engine was stop, the audio was on, the door was unlocked, the trunk was unlocked, the head lights were on, the interior light was on, the alarm was on, the GPS was on.

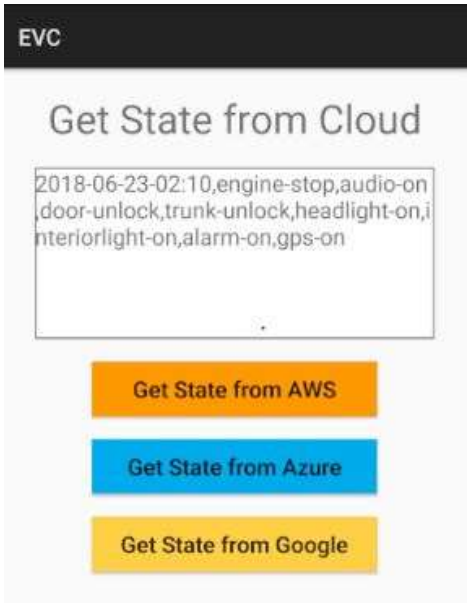


Figure 95. The vehicle state of AWS in mobile client

Figure 96. The vehicle state of Azure in mobile client

Figure 96 shows the vehicle state got from Azure, on the date “2018-06-23-05:00”, on the vehicle, the engine was stop, the audio was off, the door was locked, the trunk was locked, the head lights were off, the interior light was on, the alarm was off, the GPS was off.

Figure 97 shows the vehicle state got from Google Cloud Platform. On the date “2018-06-23-04:08”, on the vehicle, the engine was stop, the audio was off, the alarm was off, the door was unlocked, the trunk was unlocked, the head lights were off, the interior light was off, the GPS was off.

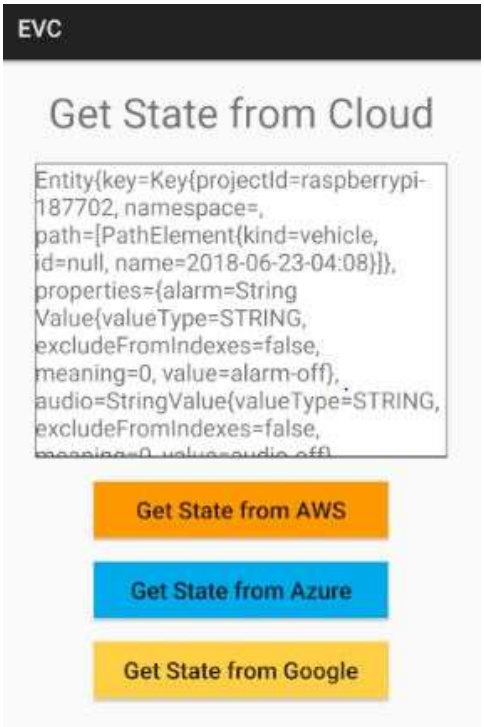


Figure 97. The vehicle state of Google Cloud in mobile client

8. Conclusion

This paper presented the design and implementation of CoT architecture based on proxy using IETF CoAP and OCF IoTivity for connectivity between IoT Networks and cloud, there are five different architectures which are able to collect massive sensing data from IoT devices and then upload the data to cloud for the further analysis. The first CoT architecture is for Hadoop and IoT platforms, IoT platform is able to upload sensing data files to Hadoop Distributed File System (HDFS). The second CoT architecture is for Hadoop and IoT Networks, there is an agent that is able to get sensing data from IoT devices and upload the data to HDFS using CoAP protocol. The third CoT architecture is for global cloud and IoT networks, AWS, Azure, and Google Cloud is used as global cloud and is able to communicate with IoT networks using MQTT protocol and store the sensing data. The fourth CoT architecture is for global cloud and IoT networks based on proxy, the proxy is able to get sensing data from IoT networks and send to store in cloud (AWS IoT, Azure IoT Hub, and Google Cloud IoT Core) using OCF IoTivity and MQTT protocol. After some experiments we also compared the IoT cloud services of AWS, Azure, and Google Cloud Platform. The fifth CoT architecture is for vehicle monitoring and control service, the client is able to monitor vehicles, control vehicles, and get vehicle state from cloud. During the design and implement of these architectures, we had more understanding about IoT services based on Clouds. In the future, we aim to make comparison by adding more IoT devices and develop a more complicated comparison system.

References

1. J.Song, A. Kunz, M. Schmidt, and P. Szczytowski. “Connecting and managing m2m devices in the future internet,” Mobile Networks and Applications, pp. 1–14, 2013.
2. Husain, Syed, et al. "Interworking architecture between oneM2M service layer and underlying networks." Globecom Workshops (GC Wkshps), 2014. IEEE, 2014.
3. Kim J, Yun J, Choi S C, et al. Standard-based IoT platforms interworking: implementation, experiences, and lessons learned[J]. IEEE Communications Magazine, 2016, 54(7): 48-54.
4. L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” Comput. Netw., vol. 54, no. 15, pp. 2787–2805, 2010.
5. D. Evans, “The internet of things how the next evolution of the internet is changing everything,” White Paper, Cisco, April 2011.
6. Jiang, Lihong, et al. "An IoT-oriented data storage framework in cloud computing platform." IEEE Transactions on Industrial Informatics 10.2 (2014): 1443-1451.
7. Derhamy, Hasan, et al. "A survey of commercial frameworks for the Internet of Things." Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on. IEEE, 2015.
8. Botta, Alessio, et al. "On the integration of cloud computing and internet of things." Future internet of things and cloud (FiCloud), 2014 international con-ference on. IEEE, 2014.
9. S. Chenishkian, “Building Smart Services for Smart Home,” in Proceedings of the IEEE 4th International Workshop on Network Appliances, pp. 215-224, 2002.
10. Kang, Byeongkwan, et al. "IoT-based monitoring system using tri-level context making model for smart home services." Consumer Electronics (ICCE), 2015 IEEE International Conference on. IEEE, 2015.
11. Sivaraman, Vijay, et al. "Network-level security and privacy control for smart-home IoT devices." Wireless and Mobile Computing, Networking and Com-munications (WiMob), 2015 IEEE 11th International Conference on. IEEE, 2015.
12. Zhang, Qi, Lu Cheng, and Raouf Boutaba. "Cloud computing: state-of-the-art and research challenges." Journal of internet services and applications 1.1 (2010): 7-18.
13. Zhou, Minqi, et al. "Services in the cloud computing era: A survey." Universal Communication Symposium (IUCS), 2010 4th International. IEEE, 2010.
14. Liu, Chang, et al. "External integrity verification for outsourced big data in cloud and IoT: A big picture." Future Generation Computer Systems 49 (2015): 58-67.
15. Soliman, Moataz, et al. "Smart home: Integrating internet of things with web services and cloud computing." Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on. Vol. 2. IEEE, 2013.
16. He, Wu, Gongjun Yan, and Li Da Xu. "Developing vehicular data cloud ser-vices in the IoT environment." IEEE Transactions on Industrial Informatics 10.2 (2014): 1587-1595.
17. Li, Ang, et al. "CloudCmp: comparing public cloud providers." Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, 2010.

18. Afonso, José A., et al. "IoT system for anytime/anywhere monitoring and control of vehicles' parameters." (2017).

19. Tahat, Ashraf, et al. "Android-based universal vehicle diagnostic and tracking system." Consumer Electronics (ISCE), 2012 IEEE 16th International Symposium on. IEEE, 2012.

20. Yao, Leehter, Yu-Qiao Chen, and Wei Hong Lim. "Internet of things for electric vehicle: An improved decentralized charging scheme." Data Science and Data Intensive Systems (DSDIS), 2015 IEEE International Conference on. IEEE, 2015.

21. Monteiro, Vítor, et al. "A flexible infrastructure for dynamic power control of electric vehicle battery chargers." IEEE Transactions on Vehicular Technology 65.6 (2016): 4535-4547.

22. Aazam, Mohammad, et al. "Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved." Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on. IEEE, 2014.

23. Yu, Zhang, et al. "Optimization design method of communication service system for vehicle remote monitoring based on Netty pattern." Chinese Automation Congress (CAC), 2017. IEEE, 2017.

24. Al-Taee, Majid A., Omar B. Khader, and Nabeel A. Al-Saber. "Remote monitoring of vehicle diagnostics and location using a smart box with Global Positioning System and General Packet Radio Service." Computer Systems and Applications, 2007. AICCSA'07. IEEE/ACS International Conference on. IEEE, 2007.

25. Thishone, P., and J. Samson Isaac. "Development of remote vehicle monitoring system for surveillance applications." Innovations in Electrical, Electronics, Instrumentation and Media Technology (ICEEIMT), 2017 International Conference on. IEEE, 2017.

26. Jun, Xu, and Liu Zhou. "Lithium battery remote monitoring system for vehicle mounted." Control And Decision Conference (CCDC), 2017 29th Chinese. IEEE, 2017.

27. Distefano, Salvatore, Giovanni Merlino, and Antonio Puliafito. "Enabling the cloud of things." Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on. IEEE, 2012.

28. Petrolo, Riccardo, Valeria Loscri, and Nathalie Mitton. "Towards a smart city based on cloud of things." Proceedings of the 2014 ACM international workshop on Wireless and mobile technologies for smart cities. ACM, 2014.

29. Tei, Kenji, and Levent Gurgen. "ClouT: Cloud of things for empowering the citizen clout in smart cities." Internet of Things (WF-IoT), 2014 IEEE World Forum on. IEEE, 2014.

30. Aazam, Mohammad, Pham Phuoc Hung, and Eui-Nam Huh. "Smart gateway based communication for cloud of things." Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on. IEEE, 2014.

31. Petrolo, Riccardo, et al. "The design of the gateway for the cloud of things." Annals of Telecommunications 72.1-2 (2017): 31-40.

32. Bhattasali, Tapalina, Rituparna Chaki, and Nabendu Chaki. "Secure and trusted cloud of things." India Conference (INDICON), 2013 Annual IEEE. IEEE, 2013.

33. Amazon Web Service, aws.amazon.com

34. Bermudez, Ignacio, et al. "Exploring the cloud from passive measurements: The Amazon AWS case." INFOCOM, 2013 Proceedings IEEE. IEEE, 2013.

35. Bracci, Fabio, Antonio Corradi, and Luca Foschini. "Database security management for healthcare SaaS in the Amazon AWS Cloud." Computers and Communications (ISCC), 2012 IEEE Symposium on. IEEE, 2012.

36. Microsoft Azure, azure.microsoft.com

37. Viswanathan, Lalitha, et al. "Predictive Provisioning: Efficiently Anticipating Usage in Azure SQL Database." Data Engineering (ICDE), 2017 IEEE 33rd International Conference on. IEEE, 2017.

38. Forsström, Stefan, and Ulf Jennehag. "A performance and cost evaluation of combining OPC-UA and Microsoft Azure IoT Hub into an industrial Internet-of-Things system." Global Internet of Things Summit (GIoTS), 2017. IEEE, 2017.

39. Google Cloud Platform, cloud.google.com

40. Bunch, Chris, et al. "An evaluation of distributed datastores using the AppScale cloud platform." Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on. IEEE, 2010.

41. Mishra, Asit K., et al. "Towards characterizing cloud backend workloads: insights from Google compute clusters." ACM SIGMETRICS Performance Evaluation Review 37.4 (2010): 34-41.

42. Tom White. Hadoop: The Definitive Guide. O'Reilly Media, Inc. 2009

43. Chuck Lam. Hadoop in Action. 2010-12-22

44. https://en.wikipedia.org/wiki/Representational_state_transfer

45. http://coap.technology/

46. IoTivity, www.iotivity.org

47. MQTT, http://mqtt.org/faq