SDN-Based Management Framework for IoT

Chin-Shiuh Shieh, Jhih-Ying Yan, and Hao-Xiang Gu

***Abstract*—Internet of Things (IoT) is getting more and more popular. This trend brings about great challenge in system management for administrators. A framework featuring Software-Defined Networking (SDN) and Message Queuing Telemetry Transport (MQTT) is proposed in this study to facilitate the deployment and management of IoT. With the proposed framework, the network connectivity and routing of a newly added node can be automatically configured. The functionalities, including data acquisition and device configuration, of an individual end device can be remotely managed via friendly web-bases user interface. Flexibility in deployment, sophistication in management and failover of faulty route can be achieved with the proposed framework.**

***Index Terms*—IoT, SDN, MQTT.**

I. Introduction

The Internet has dramatically changed our daily life. The Internet of Things (IoT) [1] will be another wave of revolution, in particular with the rise of the concept of Industry 4.0. In the era of IoT, it is desirable to endow conventional devices with communication capability. It can be expected that there will a vast number of devices to be networked. This means that networking management will be a major issue in the application and deployment of IoT. To this problem, we proposed a management framework incorporating MQTT (Message Queuing Telemetry Transport) [2] and SDN (Software-Defined Networking) [3]. MQTT is employed for the interoperability in message dispatch and SDN is incorporated for flexibility in device deployment and failover of faulty route.

The rest of this article is organized as follows. Section II briefly reviews technologies involved in the proposed framework. System design is presented in Section III. Section IV reports our system implementation. Finally, some conclusions are drawn in Section V.

II. Related Technologies

The term "Internet of Things" was coined by Kevin Ashton, director of Auto-ID center of MIT, in 1999. He defined IoT as connecting everything to Internet with communication unit in order to achieve intelligent identification and management. The "ITU Internet Report 2005: Internet of Things" is an important milestone for the evolution of IoT. European Telecommunications Standards Institute (ETSI) suggests that a complete IoT architecture should consist of 3 layers, namely perception layer, network layer, and application layer.

A. Message Queuing Telemetry Transport

Numerous protocols have been proposed for M2M (Machine-to-Machine) message exchange, such as MQTT, CoAP (Constrained Application Protocol) and XMPP (Extensible Messaging and Presence Protocol).

MQTT is used in our framework for several reasons, such as many-to-many transmission, light-weighted, and QoS support. Based on TCP/IP, MQTT aims to offer reliable service in unreliable, low-bandwidth network environment. Three different roles are involved in a MQTT system, namely publisher, broker, and subscriber, as shown in Fig. 1. Publishers are information sources. They send messages to the broker if there is any status change. Broker caches messages from publishers and send them to subscribed subscribers. There are a number of broker implementations, such as Mosquitto, Apache Apollo, HiveMQ, Mosca, RabbitMQ, and RSMB and so on. Mosquitto is employed in our implementation because of it is open-source, high-performance, and with more functional supports.

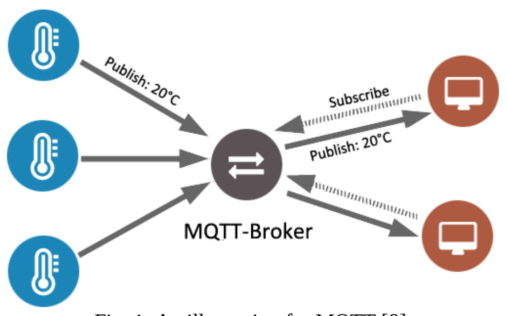


Fig. 1. An illustration for MQTT [8].

Depending on the network environment, MQTT offers 3 types of QoS services, namely at most once, at least once and exactly once. This allows the system has more control over the message dispatch.

B. JavaScript Object Notation

As IoT getting popular, there are more and more heterogeneous devices deployed in enterprises, households and factories. These devices generate and transmit data at different time and at different rate. The transmitted data are in certain formats. The formats are defined by manufactures or alliances. There will be interoperability issue if the data are in different formats. It is critical to have a common data specification for message exchange. For the time being, there are two major languages for data exchange, namely XML (eXtensible Markup Language) and JSON (JavaScript Object Notation) [4]. XML has more complete definition on data format. As a result, XML has larger message size and

Manuscript received October 25, 2018; revised January 12, 2019. This work was supported in part by the Ministry of Science and Technology, Taiwan, ROC, under MOST 106-2221-E-151-007.

Chin-Shiuh Shieh, Jhih-Ying Yan, and Hao-Xiang Gu are with the Department of Electronic Engineering, National Kaohsiung University of Science and Technology, Taiwan, ROC (e-mail: csshieh@ nkust.edu.tw).

therefore slower parsing and processing speed. On the other hand, JSON has smaller message size, which can speed up the processing. There reasons make JSON more suitable for networked applications. So we adopt JSON in our implementation.

C. Embedded Systems

Embedded systems are those appliances or controllers with embedded specific and real-time functionalities. From the perspective of applications, IEEE defines embedded systems as apparatuses controlling, monitoring or assisting the operation of equipment, machines and factories. Embedded system is one of the main reasons for the prosperity of IoT. The rise of IoT is driven by the rapid grow of embedded systems and their demands in connecting to the Internet. Embedded systems are usually deployed in household, factory automation, vehicular, medical and military applications. In recent years, a number of platforms are introduced in response to the wide variety of application needs, such as Arduino and Raspberry Pi. We shall use them as the sensor nodes/IoT device in out IoT environment.

D. Asynchronous JavaScript and XML

Asynchronous JavaScript and XML (AJAX) [5] is a browser-end web-page development technology. The primary appeal of AJAX is that part of the web page content can be updated without the need of reloading the entire web page. AJAX consists of 3 technologies, namely HyperText Markup Language (HTML)/eXtensible HyperText Markup Language (XHTML), Document Object Model (DOM) and JavaScript. HTML/XHTML is in charge of the presentation of the web page. DOM is the tool for the dynamic download of part of the web page content. JavaScript is used to implement the AJAX engine. AJAX engine is responsible for the communication between client and server. It takes care of asynchronous incoming requests. In another word, users can continue her/his browsing operation without waiting for the response from server. AJAX is widely used in the industry, such as Google Map and Yahoo News. In out implementation of the proposed framework, administrators of an IoT environment configure and monitor end devices through browser. So we also incorporate AJAX in our implementation.

E. Distributed Non-Relational Databases

NoSQL [6] is a new type of database, different from conventional relational databases. It does not use Structured Query Language (SQL) as its query language. Its storage needs not to be in table mode. It in general avoids the JOIN operation in SQL databases and has the feature of horizontal extensibility. NoSQL can conduct horizontal expansion in order to have new server node added. There is no need for high-performance server or cluster as in conventional relational database. NoSQL distributes and copies data to individual nodes and synchronize them.

NoSQL databases adopt Key-Value model to resolve the difficulty in database update for huge amount of data. Moreover, object-based APIs allow network manager easily have access to data structures stored in memory.

Big data shall accompany the growth of IoT. In this situation, NoSQL database is superior to conventional relational SQL database. Although there will be limited data in our prototype, for reason of completeness, we employ NoSQL databases in our implementation.

F. Software-Defined Networking

Software-Defined Networking (SDN), as a new technology for network virtualization, had received considerable attention since 2014. As the name implies, in SDN, the network topology and packet routing are governed by software. According to Open Networking Foundation [7], the fundamental idea of SDN is the separation of control plan and data plan, as shown in Fig. 2.

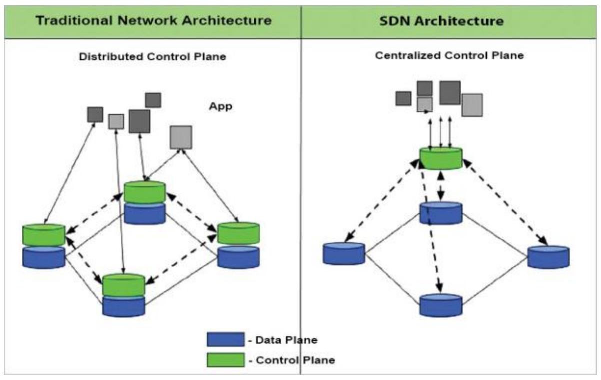

Traditional Network Architecture
SDN Architecture
Distributed Control Plane
Centralized Control Plane
App
- Data Plane
- Control Plane

Fig. 2. Separation of control and data plans in SDN [9].

In conventional networks, routers and switches operate according to their individual software and hardware configurations. Control and data plans are mixed and fused together. It is extremely difficult to integrate and coordinate these routing devices. On the other hand, in SDN, control plan is separated from data plan and SDN controller is introduced to control the forwarding behaviors of all switches in the same domain. Switches are now merely responsible for the packet forwarding.

With SDN, when network topology change is needed, there will be no more tangling of network cables. The administrator simply commands the controller to send adequate forwarding rules to individual switches. She/he needs not to configure all the routers and switches one by one. As a result, high level of flexibility and quick deployment can be achieved by SDN.

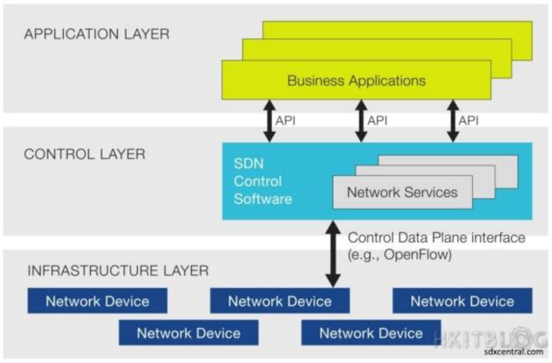


Fig. 3. Layered structure of SDN [10].

In SDN, all network devices are controlled by a SDN controller in a centralized manner. SDN controller is responsible for the maintenance of network structure and provides useful APIs (Application Programming Interface) to upper layer. With these APIs, upper layer applications can monitor and control the entire network, as illustrated in Fig. 3.Various applications become possible, such as network security, virtual segmentation, load balancing, QoS support, and so forth.

As cloud computing and big data technologies getting mature, there is increasing demands on networking supports. The industry anticipates the incorporation of SDN can get rid of the bondage of conventional networking and provide desired flexibility in network management.

III. System Design

The proposed framework aims to liberate administrators from the tangling of wired cables, and enable them to manage the network easily, flexibly and efficiently. A typical scenario is given in Fig. 4. IoT devices send two types of information to system databases. One is the device information which will be stored in MySQL database. The other is sensed data which will be stored in MongoDB. Device information includes device ID and its capability. This information is published to the broker via MQTT protocol. At the same time, a device also subscribes to its own ID in order to accept commands from users. Using browser as user-interface, users can search the databases and subscribe to interested topics. Users can also command end devices via browser. For instance, users can instruct an end device to change its updating interval.

SDN plays essential role in the proposed framework in routing configuration, traffic monitoring and failover of faculty route.

Table I summarizes the accomplishments of the proposed framework.

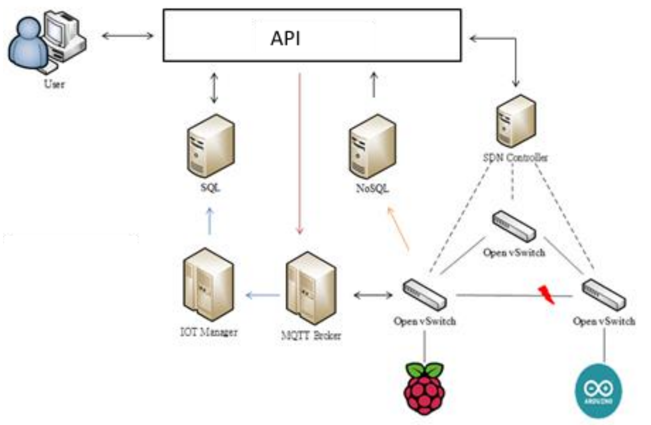


Fig. 4. A typical scenario of the proposed framework.

A. Signaling Charts for Network Topology Management

There are 3 main tasks in SDN-based network management. Here we look into their signaling diagrams one by one. The first task is the inclusion of a newly added IoT device. A new IoT device publishes its own information in JSON format to the broker via MQTT protocol. All packets during the interaction are managed by SDN, as show in Fig. 5. IoT Manager governs all registered IoT devices in the environment, and store device information and sensed data into databases. Users can then have access to sensed data via browser.

TABLE I: Accomplishments of the Proposed Framework

Technology	Function	Description
SDN	Flexible Route Configuration	Switch packet-in unmatched packet to controller for route configuration. Controller obtains detailed information regarding the packet via OFPPacketIn and then conducts flooding. Once target address is reached, add_flow is called to have a new entry added to the forwarding table in switch.
	Traffic Monitoring	Via _monitor API, controller periodically requests traffic statistics from registered switches in the network domain.
	Faulty Route Failover	Switches exchange information via BPDU packet. Root switch report exchanged information to controller. In case of faulty route, spanning tree algorithm is executed to find out a new connected topology.
MQTT	QoS Management	There are 3 types of QoS supports, namely at most once, at least once and exactly once.
	End Device Management	Control the interval, QoS and status of end devices' updating.
SQL/ NoSQL	Information/ Data Storage	SQL stores device information and NoSQL store sensed data.
AJAX	Dynamic Web Page Access	Part of web page content can be updated without the need of reloading an entire web page.
Embedded Systems	End Devices	Arduino and Raspberry Pi serve as IoT end devices in out implementation.
JSON	Data Exchange	Specification language for data exchange.

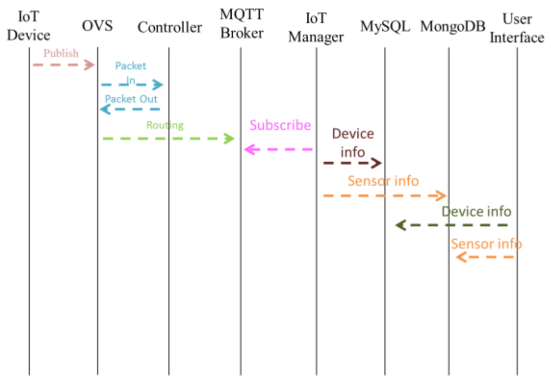


Fig. 5. Signaling diagram for the inclusion of a newly added device.

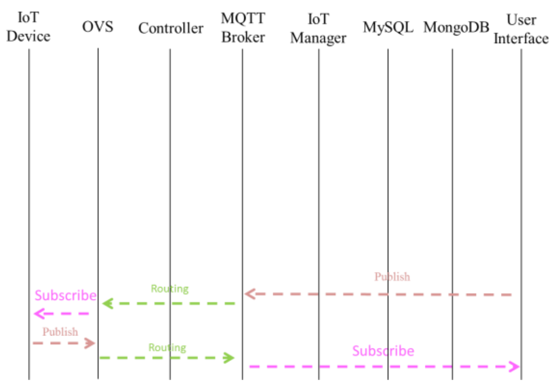


Fig. 6. Signaling diagram for device reconfiguration.

The second task is the issuing of users' commands. User can not only read sensed data, but also reconfigure end devices. From SQL database, users can have information regarding the device of interest, such as device ID, sensor type, report interval, QoS support and so on. Users can then reconfigure the device by publishing intended reconfiguration to the broker. End device receives these reconfiguration commands by subscribing to its own ID, as shown in Fig. 6. With this approach, users need not to attend the field to conduct on-site reconfiguration. Moreover, MQTT is also a good solution to the interoperability problem.

A final task is the handling of faulty route. In SDN, switches exchange status and routing information via NPDU (Bridge Protocol Data Unit) packets. Root switch passes collected information to the controller. In case of faulty route, spanning tree algorithm is invoked by controller to find out a new connected topology, as show in Fig. 7.

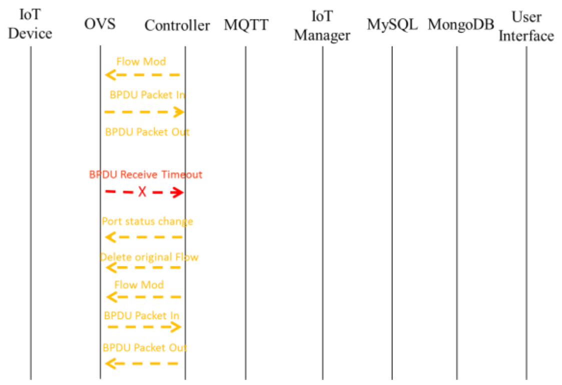


Fig. 7. Signaling diagram for failover of faulty route.

IV. Implementation

A fully functional prototype is implemented to validate the feasibility of the proposed framework. Software/hardware used in our implementation is listed in Table II.

TABLE II: Software/Hardware Used in Implementation

Item	OS	Name	Host	Qty
OpenFlow Controller	Ubuntu 14.04	Ryu 3.27	MSI Cubi	1
OpenFlow Switch	Ubuntu 14.04	Open vSwitch 2.3.0	MSI Cubi	3
MQTT Broker	Ubuntu 14.04	Mosquitto	MSI Cubi	1
USB Adaptor	N/A	RJ45	N/A	7
WiFi AP	N/A	RT-AC66U	N/A	1
IoT Manager/ SQL	Ubuntu 14.04	N/A	ASUS D810	1
NoSQL	CentOS	MongoDB	AcerPower M6	4

A. *Automatic Routing Configuration*

As an end device added to the system, its packet routing path will be automatically configured by SDN. At first, there will be no matched entry in the switch for the newly added device. The switch packet_in its packet to the controller. Controller will check if the MAC address had been recorded. If not, packet_out will be called to look for the destination MAC address by flooding. Once found, a new forwarding rule for the MAC address will be added by calling add_flow. The route configuration is then complete, as shown in Fig. 8, and subsequent packets shall follow the same routing path.

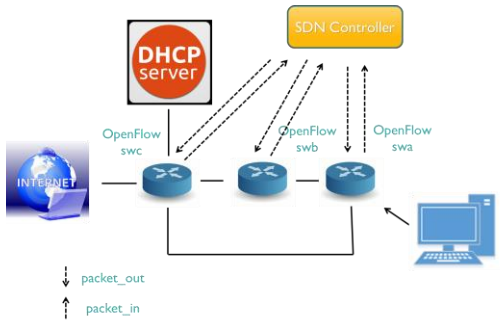


Fig. 8. Automatic routing configuration.

B. *Traffic Monitoring*

After registration, controller request information from registered switches every 10 seconds. Available information, including port, flow entry table, number of received packets (rx-pkts), number of received bytes (rx-bytes), number of receiving errors (rx-error), number of transmitted packets (tx-pkts), number of transmitted bytes (tx-bytes), number of transmitting errors (tx-error) and so on.

C. *Failover of Faulty Route*

Here we experiment the failover capability of SDN. Referring to Fig. 9, end device has an initial route following SWa/Port6 → SWa/Port8 → SWc/Port1 → SWc/Port3 → Internet. When we deliberately disable the link from SWa/Port8 to SWc/Port1, the failover mechanism invoked automatically to find out a new route. A new route following SWa/Port6 → SWa/Port7 → SWb/Port3 → SWb/Port2 → SWc/Port2 → SWc/Port3 → Internet is established and the completes the failover process.

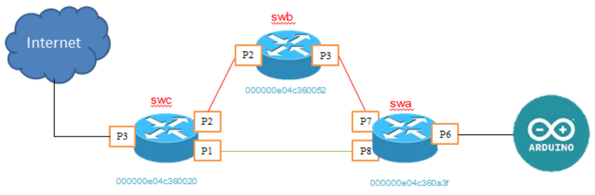
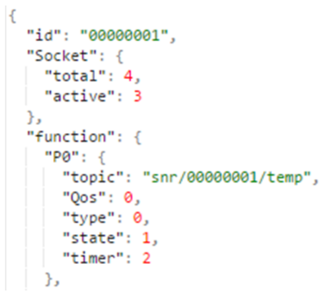


Fig. 9. Failover of faculty route.

D. *Prototype in Action*


```
{
  "id": "00000001",
  "Socket": {
    "total": 4,
    "active": 3
  },
  "function": {
    "P0": {
      "topic": "snr/00000001/temp",
      "Qos": 0,
      "type": 0,
      "state": 1,
      "timer": 2
    },
```
Fig. 10. JSON for data exchange.

For the purpose of demonstration, we setup an SDN-based management framework for IoT with topology as shown in Fig. 4. End devices are Arduino and Raspberry Pi with temperature and humidity sensors. We define a JSON format, as shown in Fig. 10, for the data exchange between end device and the IoT Manager. The defined JSON includes device ID, number of port, topic, QoS, sensor type, sensor status, sensing time, and so on.

Our implementation makes use of AJAX and MQTT for the configuration of end devices. The user interface is given in Fig. 11. It includes Dashboard, Chart, Control Panel and Manual. The page layout is designed using Cascading Style Sheets (CSS). For a professional outlook, packages imported include Font Awesome, Bootstrap, Morris, and SB admin.

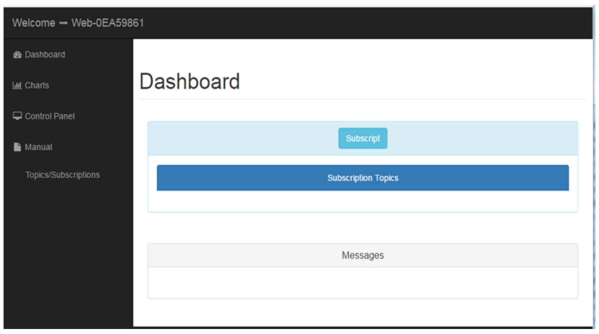
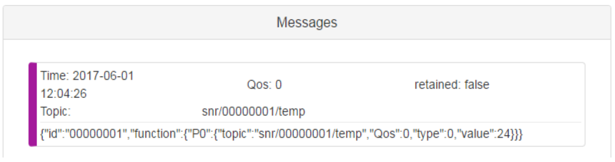


Fig. 11. Web-based user interface.

After subscription, we can see the updated message, as shown in Fig. 12.



Time: 2017-06-01 12:04:26   Qos: 0   retained: false
Topic: snr/00000001/temp

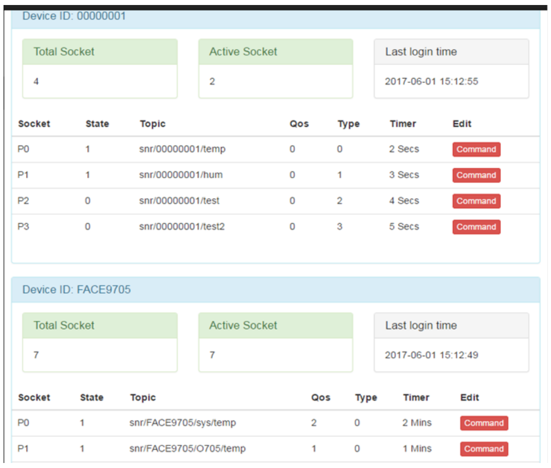
```
{"id":"00000001","function":{"P0":{"topic":"snr/00000001/temp","Qos":0,"type":0,"value":24}}}
```

Fig. 12. Received update in JSON.

User can reconfigure end devices via the web-based user interface. “Command” button allows users to specify what parameters to be updated, including QoS, On/Off control, sensing interval, topic, and so on. Click “Action” will commit the reconfiguration and publish it to the broker, as shown in Fig. 13.

V. Conclusion

A SDN-based management framework for IoT is proposed in this article. The proposal is aimed to relieve the burden of administrators in network management. A fully functional prototype indicates that the proposed framework is feasible. With SDN, the routing of newly Added devices can be automatically configured. Failover of faulty route can be achieved transparently. With the incorporation of JSON, MQTT and AJAX, users can have access to sensed data through friendly web-based user interface. Moreover, users are capable of reconfiguring end devices remotely, without the need of on-site reconfiguration.



Device ID: 00000001

Total Socket	Active Socket	Last login time
4	2	2017-06-01 15:12:55

Socket	State	Topic	Qos	Type	Timer	Edit
P0	1	snr/00000001/temp	0	0	2 Secs	Command
P1	1	snr/00000001/hum	0	1	3 Secs	Command
P2	0	snr/00000001/test	0	2	4 Secs	Command
P3	0	snr/00000001/test2	0	3	5 Secs	Command

Device ID: FACE9705

Total Socket	Active Socket	Last login time
7	7	2017-06-01 15:12:49

Socket	State	Topic	Qos	Type	Timer	Edit
P0	1	snr/FACE9705/sys/temp	2	0	2 Mins	Command
P1	1	snr/FACE9705/O705/temp	1	0	1 Mins	Command

Fig. 13. End device reconfiguration.

References

[1] M. Kaufmann, *Internet of Things: Principles and Paradigms*, R. Buyya and A. V. Dastjerdi, Eds. 2016.

[2] MQTT. [Online]. Available: https://en.wikipedia.org/wiki/MQTT

[3] F. Hu, *Network Innovation through OpenFlow and SDN: Principles and Design*, CRC Press, 2014.

[4] JSON. [Online]. Available: https://www.json.org

[5] AJAX. [Online]. Available: https://en.wikipedia.org/wiki/Ajax_(programming)

[6] S. Tiwari, *Professional NoSQL*, Wrox, 2011.

[7] Open Networking Foundation. [Online]. Available: https://www.opennetworking.org

[8] Sami Pietikäinen. Using local MQTT broker for cloud and interprocess communication. [Online]. Available: https://pagefault.blog

[9] Jitendra Bhati. A primer on software defined networking (SDN) and OpenFlow standard. [Online]. Available: https://opensourceforu.com

[10] Scott Fulton III, What is SDN? How software-defined networking changed everything. [Online]. Available: https://www.zdnet.com

Chin-Shiuh Shieh received his Ph.D. degree from Department of Computer Science and Engineering, National Sun Yat-Sen University, Taiwan, in 2009. He is now serves as an associate professor in Department of Electronic Engineering, National Kaohsiung University of Science and Technology, Taiwan. His research interests include computer networking, embedded system and computational intelligence.

Jhih-Ying Yan received his M.S. degree from Department of Electronic Engineering, National Kaohsiung University of Science and Technology, Taiwan, in 2017. His research interests include computer networking and IoT.

Hao-Xiang Gu received his M.S. degree from Department of Electronic Engineering, National Kaohsiung University of Science and Technology, Taiwan, in 2017. His research interests include computer networking and SDN.