Design Issues for Communication Protocols Conversion Scheme of IoT Devices

Shin-Jer Yang, Ting-Chen Wei

Dept. of Computer Science and Information Management, Soochow University, Taipei, Taiwan
sjyang@csim.scu.edu.tw, wtbmop@gmail.com

Abstract

Despite IoT technology provides an open and distributed networking environment, there is no compulsory standard of communication protocol for the description of object information. As a result, IoT devices with different communication protocols would require relevant converters to perform semantic analysis, or else communication between IoT devices may fail. Therefore, this paper proposes to design the conversion scheme for communication protocols of IoT devices (IoT-CPCS). This proposed scheme aims to integrate the formats of the data collected by different IoT devices, convert these data into useful and important information, present the converted information in readable message formats, and consequently store these messages in virtual servers built in the cloud platform. In addition, we conduct simulation experiments on IoT-CPCS and original MQTT transmission mechanism called MQTT Broker, and evaluate them with three key performance indicators (KPIs) such as av*er*age conversion time, system throughput, and average latency time. The simulation results indicate that the improving ratios for using the IoT-CPCS in average conversion time are 22.7%, 26.2%, 16.8%, 13.7%, 16.7% under 300, 500, 100, 2000, and 4000 packets respectively, and the system throughput is improved by 58.9% in average. Although IoT-CPCS in average latency time is a little poor than the MQTT Broker, but the latency ratio may not exceed 10%. The proposed IoT-CPCS scheme can improve the application compatibility of IoT for facilitating wider promotion, and raise the applicability level of IoT. In the future, we will consider another common communication protocols used in the IoT such as AMQP, and accuracy ratio of other KPI to evaluate the IoT-CPCS scheme with a larger amount of data.

Keywords: Cloud computing, IoT-CPCS, MQTT Broker, CoAP, Modbus-TCP

1 Introduction

In 1995, Bill Gates mentioned the concept of connection of objects in “The Road Ahead”. At that time, this concept could neither be achieved nor attract public attention because of the limitation on wireless networking and hardware as well as various sensing devices. As technology continues to advance over years, the International Telecommunication Union formally proposed the concept of IoT in 2005 [1] and therefore IoT has gained the attention from all sectors. The IoT is used to connect millions and billions of devices to communicate and share the information to all users [2].

With the active development of many equipment manufacturers, various IoT-related devices and applications have been explored, such as smart home, smart transportation, smart medicine, etc. [3], as well as smart city using sensors and smart information processing systems for managing the daily traffic in city [4]. Also, home energy management systems assist users to effectively know electronic appliance usage [5]. In addition, Cloud computing offers computing resources that delivered as a service across the entire local net or the Internet [6]. However, there is no standard of communication protocol for the description of object information and format nowadays such that the description of the information by sensor objects may differ from vendor to vendor. Therefore, it is impossible for various IoT devices to understand each other’s information contents. Therefore, this paper proposes to design a communication protocols conversion scheme for IoT devices (IoT-CPCS). This proposed scheme aims to integrate the formats of the data collected by different IoT devices, convert these data into useful and essential information by performing a specific communication protocol and semantic analysis, present the converted information in readable message formats, and consequently store these messages in virtual servers built in the cloud platform. In this way, the communication between IoT devices from different vendors will no longer be restricted by communication protocols, which facilitate the subsequent development of related IoT applications and services as well as decrease the cost of semantic conversion and simplifies the steps of semantic conversion.

*Corresponding Author: Shin-Jer Yang; E-mail: sjyang@csim.scu.edu.tw
DOI: 10.3966/160792642021052203015

In summary, the main purposes and the content of this paper are described as follows:

(1) The communication protocols used in different fields of IoT applications are various. As a result, these protocols need to be semantically analyzed through a specific converter, or communication between IoT devices cannot carry out. Therefore, this paper proposes the communication protocols conversion scheme for IoT devices.

(2) This paper proposes and designs a communication protocols conversion scheme for IoT devices, i.e., IoT-CPCS. This scheme provides the function of semantic conversion by extracting the essences of various communication protocols, converts collected data into useful and essential information, presents the converted information in readable message formats, and consequently stores these messages in virtual servers built in the cloud platform.

(3) The key performance indicators (KPIs) such as the average conversion time, average latency time, and system throughput are listed. These KPIs are to compare and evaluate between IoT-CPCS and original MQTT transmission mechanism called MQTT Broker.

(4) Finally, we will perform simulations to prove that the proposed IoT-CPCS can obtain better performance based on above three KPIs. Hence, the IoT-CPCS can solve the communication problems between IoT applications in different fields, and integrates them to achieve the goals of complete effect and substantial benefit.

The remainder of this paper is organized as follows. Section 1 presents research background, research motivation and research purpose. The Section 2 is the literature on the communication protocols commonly used in the field of IoT applications. Section 3 introduces the operation flowchart and algorithm design for the IoT-CPCS scheme. Section 4 presents the simulation experimental environments and results analysis, including the simulation environments, the description of simulation process operations, the definition of KPIs, and the analysis of the experimental results. Section 5 draws a conclusion for elaborating research results and examining contributions of this paper as well as future research directions.

2 Related Work

This paper investigates three communication protocols commonly used in the IoT application field, namely MQTT, CoAP, and Modbus-TCP. Figure 1 shows the protocol stack of the IoT system [7]. It is obvious from Figure 1 that the transmissions of the three communication protocols mentioned above are based on TCP or UDP. These communication protocols can be described as following in detail.

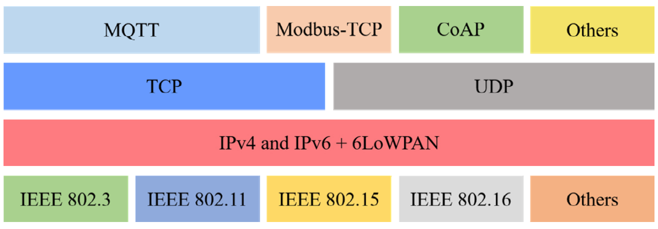

Figure 1. Protocol stack of the IoT system

2.1 MQTT

MQTT (Message Queuing Telemetry Transport) is a communication protocol developed by IBM and Eurotech. According to the introduction of the official website, MQTT is a protocol designed especially for IoT [8]. In 2003, IBM handed over MQTT to OASIS (The Organization for the Advancement of Structured Information Standards) for standardization of the protocol. Since IBM has always considered protocols as open sources for prevalence purpose, more and more people use MQTT in various fields gradually. The MQTT uses Publish/Subscribe and Broker's message transmission mechanism. Publisher, the source of the message, sends a message to Broker with its topic. Subscriber registers with Broker to request for the topic of the message. When the Publisher sends a message with its topic to Broker, all Subscribers that register with Broker will receive this message. The diagram of MQTT message transmission mechanism is shown in Figure 2.

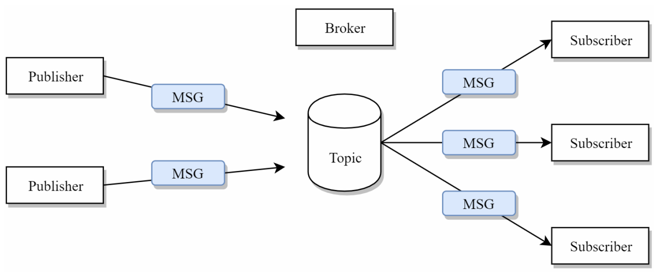

Figure 2. Diagram of MQTT message transmission mechanism

The message structure of MQTT is mainly composed of three parts. As shown in Figure 3, the message structure consists of the Fixed Header, the Variable Header, and Payload. Fixed Header is of 2 bytes. The first byte contains Message Type, DUP, QoS Level, and Retain. The second byte is Remaining Length, which includes the Variable Header and Payload, and can be extended up to 4 bytes.

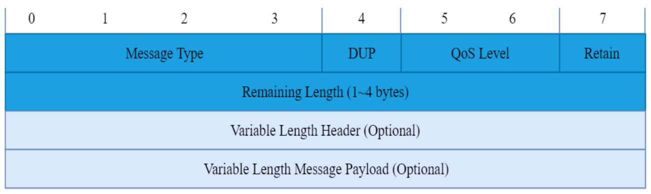
0
1
2
3
4
5
6
7
Message Type
DUP
QoS Level
Retain
Remaining Length (1~4 bytes)
Variable Length Header (Optional)
Variable Length Message Payload (Optional)

Figure 3. Message structure of MQTT

There are many examples of systems that often use MQTT in IoT applications such as the Nursing Home Patient Monitoring System [9], Web-Based IoT Solution for Monitoring Data [10], and the MQTT Based Secured Home Automation System [11]. The Nursing Home Patient Monitoring System is one of the IoT application cases. This system is responsible for collecting the vital sign measurements of patients and transmitting them to multiple nursing stations. Using the features of the MQTT Broker, the sensor data is published to the MQTT Broker, and the server of the nursing station obtains the subscribed sensor data from the MQTT Broker according to the topic of interest. This system uses MQTT in the transmission process. If the communication protocols commonly used in various fields of IoT applications can be integrated, with the characteristics of the MQTT Broker, the application level of the IoT can be improved.

2.2 CoAP

At present, this world is composed of personal computers. Message exchange is realized through TCP and HTTP (the application layer protocol). It has the feature that data transmission is confirmed through Client/Server. Therefore, it has high reliability. However, it is an excessive requirement for IoT devices to implement the TCP and HTTP protocols because the confirmation of data transmission will consume more hardware resources. Many devices in the wireless IoT are resource-constrained. These devices have a small amount of internal storage capacity and limited computation ability only. In order to allow these devices to access the IoT networking, the work team of CoRE (Constrained RESTful Environment) in IETF (Internet Engineering Task Force) proposed an application layer protocol based on REST architecture, which is the Constrained Application Protocol (CoAP), it is one of the basic desires for communication among various physical devices [12].

The CoAP is an application layer protocol that runs on the UDP protocol. Its message structure is shown in Figure 4, which consists of Ver, T, TKL, Code, Message ID, Token, Options, Payload Marker, and Payload. The CoAP protocol is very small in size, and the smallest data packet occupies four bytes only. Therefore, it is indeed a better solution to small devices like IoT devices [13]. CoAP uses the same request-response work mode as HTTP. There are four different message types in CoAP [14], namely CON, NON, ACK and RST.

2.3 Modbus-TCP

Modbus is a communication protocol for the controller of the application layer, which is used mainly for monitoring and managing devices in a master/slave mode [15]. It was originally developed by Modicon in 1979 and can be roughly divided into

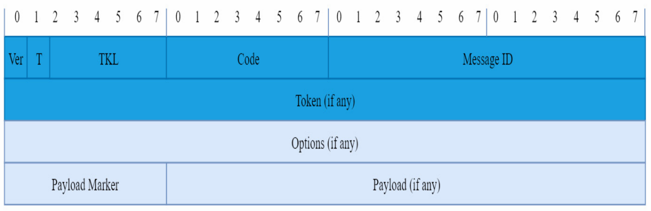

Figure 4. Message structure of CoAP

Modbus-RTU, Modbus ASCII, and Modbus-TCP. Modbus-TCP uses Ethernet TCP/IP to transfer data, which is suitable for industrial control systems in the IoT application field.

Modbus-TCP provides operation services for Client/ Server Communication Models in a Master/Slave Communication Structure as well as specific function codes [16]. That is, there is one master only and 247 slaves in the network. Only the master can send a request message to start the communication.

The message structure of Modbus-TCP is shown in Figure 5. A whole message is called ADU (Application Data Unit), which contains MBAP (Modbus Application Protocol) Header and PDU (Protocol Data Unit) [16]. MBAP Header is composed of Transaction ID, Protocol ID, Length, and Unit ID while PDU is composed of Function Code and Data.

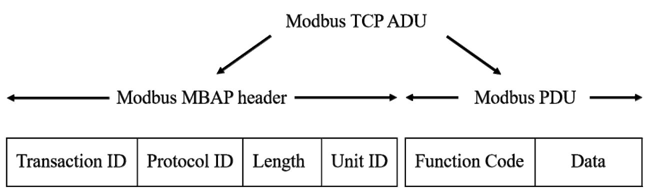

Figure 5. Message structure of Modbus-TCP

2.4 JSON

JSON (JavaScript Object Notation) was conceived and designed by Douglas Crockford. It is a lightweight data exchange language based on text, which is easy to read [17]. JSON uses not only a completely language-independent text format but also features similar to the C language family, which makes JSON an ideal data exchange language [18]. Therefore, we incorporate JSON into the scheme proposed in this paper and present the converted data in JSON message format. The structure of JSON can be divided into five types, namely object, array, value, string, and number.

2.5 Comparisons of IoT Communication Protocols

This paper compares in detail the IoT communication protocols introduced above and analyzes the characteristics of each communication protocol according to various criteria. The complete comparisons for these IoT communication protocols are shown in Table 1. The advantages and disadvantages of IoT communication protocols are show in Table 2.

Table 1. Summarized comparisons of IoT communication protocols

	MQTT	CoAP	Modbus-TCP
Year	1999	2010	1979
Standard	OASIS, Eclipse Foundations	IETF, Eclipse Foundation	de facto
Structure	Client/Broker	Client/Server or Client/Broker	Master/Slave
Mechanism	Publish/Subscribe	Request/Response or Publish/Subscribe	Request/Response
Sematic/Method	Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close	Get, Post, Put, Delete	Read, Write
Communication Protocol	TCP	UDP	TCP/UDP

Table 2. The advantages and disadvantages of IoT communication protocols

	MQTT	CoAP	Modbus-TCP
Advantages	Provides real-time and reliable messaging services to connected remote devices with limited bandwidth	Small devices with low power consumption, low computing and communication capabilities can interact through RESTful	Allow multiple devices to communicate on the same network
Disadvantages	Only save the latest Retain Message, which may cause the message to be lost or duplicated [19]	Due to lack of Internet infrastructure support, it is not compatible with firewalls, proxy servers and routers [20]	There are security concerns such as lack of confidentiality and data integrity
Application Fields	Data transmission and monitoring of remote equipment under low frequency or unreliable networks	Communication IP network with limited resources	Industrial remote monitoring

The MQTT protocol is the most used and preferred for simple and complex IoT implementations [21]. Since the MQTT transmission mechanism is published or subscribed, this study is based on the point of view of the patient monitoring system in the nursing home. This system is one case of any IoT applications to extract different communication protocols used in various fields of IoT applications and convert them into useful and important information. After converting it into usable and essential information, in addition to presenting it in a readable message format, this system is also build an MQTT Broker environment called Mosquitto on the cloud platform to store the converted data in this environment, which is convenient for cloud platform developers to follow-up processing and application. The IoT-CPCS mechanism is to utilize the same configuration of IoT device data and analyze the features of MQTT Broker. Hence, to verify the differences between the IoT-CPCS and the MQTT Broker in terms of KPIs will perform simulations, after the conversion of the three communication protocols: MQTT, CoAP, and Modbus-TCP.

3 Operation and Design Issues in IoT-CPCS

3.1 Operational Flow of IoT-CPCS

The IoT-CPCS proposed in this paper covers four relational procedures, as shown in Figure 6. The functions of this mechanism are to collect data from different IoT devices, convert them into useful and essential information after identification, and finally present the converted information in JSON message format and store it in the virtual server build in the cloud platform. Therefore, the four relational procedures of the IoT-CPCS scheme include MQTT_CONVERSION, CoAP_CONVERSION, Modbus-TCP_CONVERSION, and UPLOAD. Each procedure has their own feature, and the description is listed as follows.

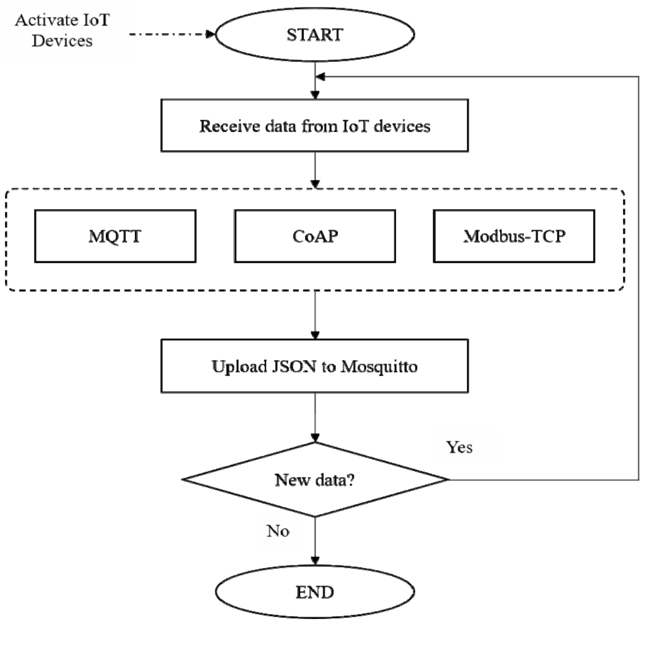

Figure 6. The operational flow cf IoT-CPCS

(1) MQTT_CONVERSION: In this procedure, MQTT data can be received. Message structures include Message Type, DUP, QoS Level, Retain, Remaining Length, Variable Length Header, and

Variable Length Message Payload. Useful and essential information including Message Type and Variable Length Message Payload can be extracted and then converted into easy-to-understand information. The comparison table before and after MQTT conversion is shown in Table 3. The message structure extracted from this procedure is the information converted by the proposed scheme, as shown in Table 4.

Table 3. The comparisons before and after MQTT conversion

MQTT	
Before	After
• Message Type • DUP • QoS Level • Retain • Remaining Length • Variable Length Header • Variable Length Message Payload	• Message Type • Variable Length Message Payload

Table 4. Message structure list of MQTT

Message Structure	Status	Descriptions
Message Type	Action	Message type of that message
Variable Length Message Payload	Topic	This structure contains the topic of the message

(2) CoAP_CONVERSION: In this procedure, CoAP data can be received. Message structures include Ver, T, TKL, Code, Message ID, Token, Options, Payload Marker, and Payload. Useful and essential information including T and Code can be extracted and then converted into easy-to-understand information. The comparison table before and after CoAP conversion is shown in Table 5. The message structure extracted from this procedure is the information converted by the proposed scheme, as shown in Table 6.

Table 5. The comparisons before and after COAP conversion

CoAP	
Before	After
• Ver • T • TKL • Code • Message ID • Token • Options • Payload Marker • Payload	• T • Code

Table 6. Message structure list of CoAP

Message Structure	Status	Descriptions
T	Message Type	Message type of that message
Code	Action	Presentation formats of Request/ Response are different.

(3) Modbus-TCP_CONVERSION: In this procedure, Modbus-TCP data can be received. Message structures include Transaction ID, Protocol ID, Length, Unit ID, Function Code, and Data. Useful and essential information including Transaction ID, Unit ID, and Function Code can be extracted and then converted into easy-to-understand information. The comparison table before and after Modbus-TCP conversion is shown in Table 7. The message structure extracted from this procedure is the information converted by the proposed scheme, as shown in Table 8.

Table 7. The comparisons before and after Modbus-TCP conversion

Modbus-TCP	
Before	After
• Transaction ID • Protocol ID • Length • Unit ID • Function Code • Data	• Function Code • Unit ID • Transaction ID

Table 8. Message structure list of Modbus-TCP

Message Structure	Status	Descriptions
Transaction ID	Transaction ID	Communication ID of Modbus Request/ Response
Unit ID	Unit ID	Identification code of remote equipment (Slave)
Function Code	Action	Master informs Slave what operation to implement.

(4) UPLOAD: This module is responsible for saving data in JSON message format in a virtual server built in the cloud platform.

3.2 Algorithm Design for IoT-CPCS

According to the operational flow of the IoT-CPCS as depicted in above Figure 6, the design and description of the algorithm using pseudocode is shown as follows.

```
Algorithm IoT-CPCS(){
Input:
mqtt_list = []    # Store MQTT data
mqtt_dict = {}
coap_list = []              #Store CoAP data
coap_dict = {}
mbstcp_list = []            # Store Modbus-TCP data
mbstcp_dict = {}
Output:
  To complete conversion to useful data and response message to device and upload to Mosquitto
Method:
BEGIN{
/#Read protocol data
  InputData = sensor.load()
/#Identify the protocol type of the data
  for i in InputData:
/#Extract useful information from MQTT and store it in JSON message format
      if (InputData [i] belong to MQTT):
        MQTT_CONVERSION(InputData [i])
        break
/#Extract useful information from CoAP and store it in JSON message format
      elif (InputData [i] belong to CoAP):
        COAP_CONVERSION(InputData [i])
        break
/#Extract useful information from Modbus-TCP and store it in JSON message format
      elif (InputData [i] belong to Modbus-TCP):
        Modbus-TCP _CONVERSION(InputData [i])
        break
      else:
        Return ("Not Identifiable Message")
        GO TO BEGIN
/#Save data in JSON message format in a virtual server built in the cloud platform
  UPLOAD(mqtt_list, coap_list, mbstcp_list)
/#Identify whether communication protocol data are received or not?
  if (sensor.load() != null):
    GO TO BEGIN
  else:
    break
}END
Procedure MQTT_CONVERSION(InputData [i]){
//#Extract useful information from MQTT and present it in JSON message format
  mqtt_dict = {
    'Time' : InputData [i][time],
    'Action' : InputData [i][ Message Type],
    'Topic' : InputData [i][Variable Length Message Payload]
  }
  mqtt_list.append(mqtt_dict)
  return mqtt_list
}END
Procedure CoAP_CONVERSION(InputData [i]){
/#Extract useful information from CoAP and present it in JSON message format
  coap_dict = {
    'Time' : InputData [i][time],
    'Message Type' : InputData [i][T],
    'Action' : InputData [i][Code]
  }
  coap_list.append(coap _dict)
  return coap_list
}END
Procedure Modbus-TCP_CONVERSION(InputData [i]){
/#Extract useful information from Modbus-TCP and present it in JSON message format
    mbstcp_dict = {
        'Time' : InputData [i][time],
        'Transaction ID' : InputData [i][ Transaction ID],
        'Unit ID' : InputData [i][ Unit ID],
        'Function Code' : InputData [i][ Function Code],
    }
    mbstcp_list.append(mbstcp_dict)
    return mbstcp_list
}END
Procedure UPLOAD(mqtt_list, coap_list, mbstcp_list){
/#Save data in JSON message format of the MQTT Broker (Mosquitto) of a virtual server built in the cloud
platform
    Upload mqtt_list, coap_list, mbstcp_list to Mosquitto
}END
}END IoT-CPCS
```

4 Simulation Environments Setup and Results Analysis

4.1 Simulation Environments Design

The simulation environment is divided mainly into three steps. Step one is IoT devices, and they will be responsible for data collection from different fields and utilize the WireShark tool to capture packets of each communication protocol. Then it passes them to the next step. Step two is the IoT-CPCS, the core part of the simulation environment. This scheme will identify the type of the communication protocol based on received data, convert the received data into useful and essential information, and present it in JSON format. The last step is to upload from converted data into AWS cloud platform. These converted data through the IoT-CPCS can be transferred to store in the virtual server (i.e. to be connected to Mosquitto) built in the Amazon AWS platform. The chart of related simulation environment architecture is shown in Figure 7. The hardware and software specifications of the related physical and virtual servers are illustrated in Table 9. The descriptions of the simulation experiment tools used in this paper are shown in Table 10.

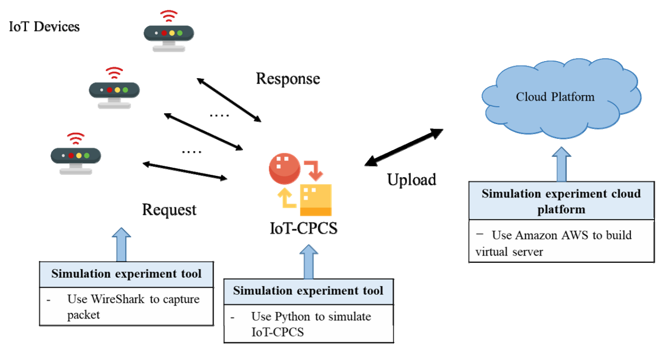
IoT Devices
Response
....
....
Request
IoT-CPCS
Upload
Cloud Platform
Simulation experiment tool
- Use WireShark to capture packet
Simulation experiment tool
- Use Python to simulate IoT-CPCS
Simulation experiment cloud platform
- Use Amazon AWS to build virtual server

Figure 7. Simulation environment architecture diagram

Table 9. The hardware and software specifications of the physical and virtual servers

Hardware / Software Configurations	Physical Server Specifications	Virtual Server Specifications
Operation System	Windows 10	Ubuntu
CPU	4 Cores	1 Core with 2.6 GHz
Memory	8GB or above	2GB or above
Disk	1TB or above	20GB or above

Table 10. The descriptions of the simulation experiment tools

Simulation Experiment Tools	Descriptions
Amazon Web Services (AWS)	AWS is a cloud computing service established by Amazon, it adopts a pay-as-you-go mechanism. Hence, the cloud users can use it according to their needs and charges.
WireShark 3.2.3	The WireShark tool is the open and free source network packet analysis software. Through the user-friendly interface operations, it can quickly retrieve the required packets.
Python 3.7	Python is a general-purpose interpreter programming language. It is easy to use and has many components, which can accomplish the task what you want easily and efficiently.

4.2 KPIs Definition and Description

Through simulation experiments, this paper compares and analyzes the IoT-CPCS and the MQTT Broker using three KPIs, i.e., average conversion time, average latency time, and system throughput. The purposes of KPIs are shown in Table 11. The calculation method of each KPI is shown in Formulas

(1), (2), and (3).

Table 11. KPIs' definitions and descriptions

KPIs	Purposes
Average Conversion Time (ACT) Unit: *ms*	It represents the average conversion time (ACT) between the receipt of the message data (*N*) from all IoT devices and the completion of conversion. The calculation method of ACT is shown in Formula (1).
Average Latency Time (ALT) Unit: ms	It is used to detect the total delay time (TDT) of all packet data in the process of semantic conversion between the packet and the packet (*N*-1) of the IoT device. The calculation method of ALT is as shown in Formula (2).
System Throughput (ST) Unit: packet	Use simulation tools to evaluate the total message data volume (*K*) of all IoT devices that actually process within a fixed period of time. That is, to identify the amount of data that can be processed within a unit of time (1000 *ms*). The calculation method of ST is shown in Formula (3).

$$\text{ACT} = \text{CT} / N \qquad \textbf{(1)}$$

$$\text{ALT} = \text{TDT} / (N - 1) \qquad \textbf{(2)}$$

$$\text{ST} = K / \text{Unit of time } (1000\ ms) \qquad \textbf{(3)}$$

4.3 Simulation Results and Analysis

In this paper, the simulation environment is set up by a physical host and a virtual host. The physical host is responsible for connecting with IoT devices to capture the packets generated during connections as well as generating the packets of the devices that cannot be simulated. Finally, the obtained packets are converted into useful and essential information through the IoT-CPCS and are presented in JSON format. The virtual host is responsible for transferring to store the converted JSON data in the virtual server built in Amazon AWS (There is a Mosquitto to be connected).

Through the connection with IoT devices and the setting of the simulation program, the amount of data received at the same time are set as 300, 500, 1000, 2000, and 4000 as well as the simulation is performed 7 rounds for each amount of data, and ignore the maximum and the minimum to obtain the average value. A simulation experiment was conducted on the MQTT Broker and the conversion scheme of the IoT-CPCS mentioned in this paper. The results prove that the improved ratios in average conversion time using the IoT-CPCS are 22.7%, 26.2%, 16.8%, 13.7%, and 16.7% and in average latency time are -8.3%, -7.7%, -6.7%, -10%, and -7.7% under 300, 500, 1000, 2000, and 4000 packets, respectively. Also, the system throughput improved by 58.5%. The average conversion time, average latency time, and system throughput will be described in detail below. Also, the simulated KPIs are summarized in Table 12.

Table 12. simulation results of KPIs

Schemes		IoT-CPCS	MQTT Broker	Improved Ratios
ACT (Unit: *ms*)	300 packets	160	207	22.7%
	500 packets	270	366	26.2%
	1000 packets	502	603	16.8%
	2000 packets	998	1156	13.7%
	4000 packets	1938	2327	16.7%
ALT (Unit: *ms*)	300 packets	13	12	-8.3%
	500 packets	14	13	-7.7%
	1000 packets	16	15	-6.7%
	2000 packets	25	22	-10%
	4000 packets	28	26	-7.7%
ST (Unit: packets)		1950	1230	58.5%

(1) Average conversion time (ACT): Calculate the conversion time during the period from receipt to response, and record them in the amount of 300, 500, 1000, 2000, and 4000 packets of data respectively. The average conversion time of the IoT-CPCS in this paper is lower than the MQTT Broker because the IoT-CPCS only extract useful and essential information, and the MQTT Broker needs to transmit a complete packet. The average conversion time is shown in Figure 8.

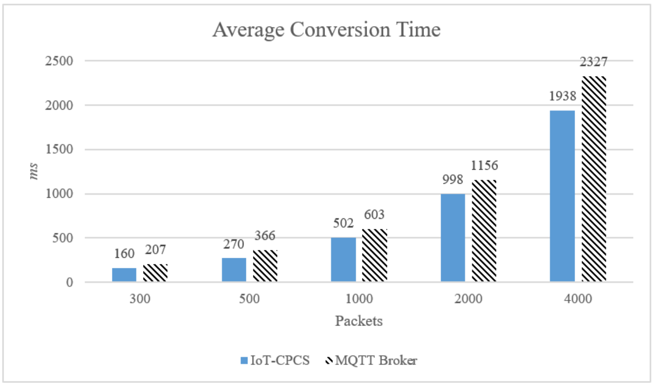

Figure 8. Comparisons of ACT between IoT-CPCS and MQTT Broker

(2) Average latency time (ALT): Record the processing time of 300, 500, 1000, 2000, and 4000 packets data for five times, subtract the average conversion time and divide by the total number of times to obtain the average value. Because the IoT-

CPCS mechanism proposed in this paper needs to retrieve available and important information and present it in a readable format, while the MQTT Broker program only needs to transmit packets, the results show that this mechanism has a higher average delay time. But the delay ratio will not exceed 10%, as shown in Figure 9.

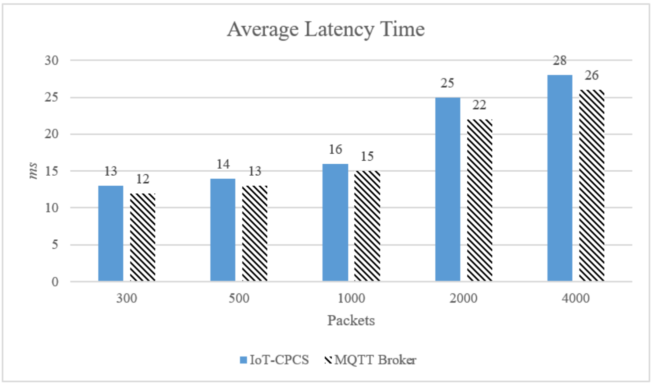

Figure 9. Comparisons of ALT between IoT-CPCS and MQTT Broker

(3) System throughput (ST): Evaluate the actual amount of data that can be processed within a unit of time period (set as 1000 milliseconds) and individually calculate the system throughput of the IoT-CPCS proposed in this paper and the MQTT Broker based on 5 simulation rounds. The results show that the amount of data that IoT-CPCS can process in a unit of time (1,950 packets) is about 1.59 times than that of the MQTT Broker (1,230 packets). The results are shown in Figure 10.

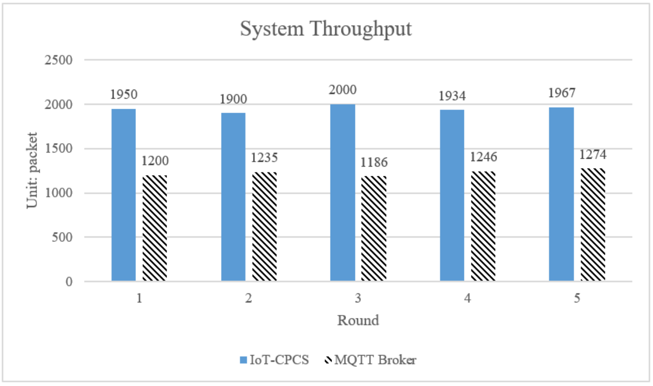

Figure 10. Comparisons of ST between IoT-CPCS and MQTT Broker

4.4 Summary of Results and Overall Analysis

In the average conversion time, the experimental results of IoT-CPCS under different data packets is lower than the MQTT Broker: the average reduction were 47 *ms*, 96 *ms*, 101 *ms*, 158 *ms*, and 389 *ms* under 300, 500, 1000, 2000 and 4000 packets. This paper analyzes the reason is the mechanism only needs to retrieve the available and important information of each communication protocol, while the MQTT Broker needs to transmit a complete packet. In the average latency time, the experimental results of IoT-CPCS under different data packets is all a little higher than the MQTT tBroker: under 300, 500, 1000, 2000 and 4000 packets, the average increase is 1 *ms*, 1 *ms*, 1 *ms*, 3 *ms* and 2 *ms*, respectively. This paper analyzes the reason is IoT-CPCS will be presented in a readable format after conversion, and the MQTT Broker only needs to transmit packets. Although the average latency time of IoT-CPCS are slightly higher than the MQTT Broker, but the latency ratio will not exceed 10%; in the system throughput, the experimental results of IoT-CPCS at a fixed interval (set to 1000 *ms*) is greater than the MQTT Broker. After five simulations, the improved ratios were 62.5%, 53.8%, 68.6%, 55.2% and 54.4%, respectively. After average, the system throughput of IoT-CPCS is about 1.59 times than the MQTT Broker. From the above simulation data, it is proved that the IoT-CPCS proposed in this paper has poor results in average latency time, but the latency ratio will not exceed 10%, and the average conversion time and system throughput of IoT-CPCS are more effective and efficient than the MQTT Broker.

5 Conclusions

In this paper, the overall research results and contributions are illustrated as follows:

(1) The communication protocols used in different IoT fields are various, which lead to the need for specific conversion of these communication protocols.

(2) The IoT-CPCS provides the function of protocols conversion. The communication protocols used in different IoT fields are extracted and converted into usable and essential information, and then be presented in a readable message format and stored on the cloud platform.

(3) This paper proposes three KPIs such as average conversion time, average latency time, and system throughput. They are compared and analyzed through simulation experiments to verify that the IoT-CPCS is more efficient than the MQTT Broker.

(4) The simulation results indicate that the improved ratios in average conversion time using the IoT-CPCS are 22.7%, 26.2%, 16.8%, 13.7%, 16.7% and also the average latency time are -8.3%, -7.7%, -6.7%, -10%, -7.7% under 300, 500, 1000, 2000, and 4000 packets, respectively. Also, the system throughput using the IoT-CPCS is improved by 58.5%, which is better than the MQTT Broker.

The IoT-CPCS can be employed in IoT applications. It can be regarded as an MQTT Broker, which can identify and process communication protocols three times more than the MQTT Broker. The Broker of the IoT-CPCS can also act as a Publisher to transmit the received message to other Brokers that need the topic of this message. In this way, the IoT-CPCS can extend to a distributed structure. This application is convenient for cloud developers to use data. They do not need to understand the message structure of each communication protocol in depth. They only need to apply or analyze the converted data. Hence, the proposed IoT-CPCS converts the data of IoT devices into useful and essential information and presents it in a readable format, is different from the general conversion. It omitted some of the characteristics of the communication protocol, and only did semantic conversion. Although the average conversion time and system throughput is better than the MQTT Broker, cloud developers who want to understand the message structure of each communication protocol need to spend more time researching because of the omission of some communication protocol's characteristics.

In the future research on the IoT-CPCS, we will add other KPIs such as accuracy ratio, and test with a larger amount of data to obtain more objective and precise results. Additionally, the collected data will be presented in real time and be used for more application development. We expect that the proposed IoT-CPCS can improve the application compatibility of IoT for facilitating wider promotion, and raise the usable level of IoT.

Acknowledgements

The authors would like to thank two anonymous reviewers for constructive comments that improve the paper.

References

[1] ITU, The Internet of Things, 2005, https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf, Retrieved on September 15, 2018.

[2] G. Ranganathan, R. Bestak, C. O. Chow, Guest Editorial: Special Issue on "Computational Approaches in Cloud Based IoT", *Journal of Internet Technology*, Vol. 21, No. 1, pp. 147-148, January, 2020.

[3] M. Soliman, T. Abiodun, T. Hamouda, J. Zhou, C.-H. Lung, Smart Home: Integrating Internet of Things with Web Services and Cloud Computing, In *Proceedings of 2013 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Bristol, UK, December, 2013, pp. 317-320.

[4] P. Sethi, S. R. Sarangi, Internet of Things: Architectures, Protocols, and Applications, *Journal of Electrical and Computer Engineering*, Vol. 2017, pp. 1-26, January, 2017.

[5] W. T. Cho, Y. W. Ma, Y. M. Huang, A smart socket-based multiple home appliance recognition approach over IoT architecture, *Journal of Internet Technology*, Vol. 16, No. 7, pp. 1227-1238, December, 2015.

[6] X. J. Chen, B. D. Chen, X. M. Jiang, X. B. Chen, W. H. Cai, Improved Cloud Computing Architecture for the Internet of Things, *Journal of Internet Technology*, Vol. 17, No. 4, pp. 683-693, July, 2016.

[7] N. Naik, Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP, In *2017 IEEE international systems engineering symposium (ISSE)*, Vienna, Austria, October, 2017, pp. 1-7.

[8] U. Hunkeler, H. L. Truong, A. Stanford-Clark, MQTT-S—A Publish/Subscribe Protocol for Wireless Sensor Networks, In *Proceedings of 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, Bangalore, India, January, 2008, pp. 1-8.

[9] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications, *Journal of IEEE Communications Surveys & Tutorials*, Vol. 17, No. 4, pp. 2347-2376, Fourth Quarter, 2015.

[10] K. Grgić, L. Špeh, I. Heđi, A Web-Based IoT Solution for Monitoring Data Using MQTT Protocol, In *Proceedings of 2016 International Conference on Smart Systems and Technologies (SST)*, Osijek, Croatia, October, 2016, pp. 249-253.

[11] Y. Upadhyay, A. Borole, D. Dileepan, MQTT Based Secured Home Automation System, In *Proceedings of 2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, Indore, India, March, 2016, pp. 1-4.

[12] J. Sidna, B. Amine, N. Abdallah, H. El Alami, Analysis and evaluation of communication Protocols for IoT Applications, In *Proceedings of the 13th International Conference on Intelligent Systems: Theories and Applications*, Rabat, Morocco, September, 2020, pp. 1-6.

[13] Z. Shelby, K. Hartke, C. Bormann, *The Constrained Application Protocol (CoAP)*, Internet Engineering Task Force (IETF), RFC7252, June, 2014.

[14] CoAP, https://coap.technology/, Retrieved on November 15, 2019.

[15] J. Kuang, G. Wang, J. Bian, A Modbus Protocol Stack Compatible with RTU/TCP Frames and Embedded Application, in: M. Zhu (Eds.), *Business, Economics, Financial Sciences, and Management*, Springer, Berlin, Heidelberg, 2012, pp. 765-770.

[16] W. Shang, Q. Qiao, M. Wan, P. Zeng, Design and Implementation of Industrial Firewall for Modbus/TCP, *Journal of Computers*, Vol. 11, No. 5, pp. 432-438, September, 2016.

[17] JSON, https://www.json.org/json-en.html, Retrieved on November 20, 2019.

[18] P. Wehner, C. Piberger, D. Göhringer, Using JSON to Manage Communication Between Services in the Internet of Things, In *Proceedings of 2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, Montpellier, France, May, 2014, pp. 1-4.

[19] C. Parian, T. Guldimann, S. Bhatia, Fooling the Master: Exploiting Weaknesses in the Modbus Protocol, *Procedia Computer Science*, Vol. 171, pp. 2453-2458, 2020.

[20] X. Wu, N. Li, Improvements of MQTT Retain Message Storage Mechanism, In *2018 2nd IEEE Advanced*

Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Xi'an, China, May, 2018, pp. 957-961.

[21] D. Glaroudis, A. Iossifides, P. Chatzimisios, Survey, comparison and research challenges of IoT application protocols for smart farming, *Computer Networks*, Vol. 168, Article No. 107037, February, 2020.

Biographies

Shin-Jer Yang is currently a full Professor in the Department of Computer Science and Information Management, Soochow University, Taipei, Taiwan. Professor Yang is the author/coauthor of more than 126 refereed technical papers (Journals and Conferences) on Wired/Wireless Networking and Applications, Cloud/Internet Computing Applications and Services, and Network Management and Security. Also, he takes in charge of more than 30 research projects. His research interests include Wired/Wireless Networking Technologies and Applications, Cloud/Internet Computing and Applications, AIoT Applications, Network Management and Security, and Information Management.

Ting-Chen Wei. Currently, She is a Teaching Assistant in the Department of Computer Science and Information Management, Soochow University, Taipei, Taiwan. Her research interests include Cloud Computing and Service, IoT Applications, and Web Applications Design.