Implementing Publish/Subscribe Pattern for CoAP in Fog Computing Environment

Jyun-Yao Huang
Department of Computer Science and Engineering
Taiwan Information Security Center at NCHU
National Chung Hsing University
Taichung, Taiwan
allen501pc@gmail.com

Po-Hsin Tsai
Department of Computer Science and Engineering
Taiwan Information Security Center at NCHU
National Chung Hsing University
Taichung, Taiwan
firmamentstar@gmail.com

I-En Liao
Department of Computer Science and Engineering
Taiwan Information Security Center at NCHU
National Chung Hsing University
Taichung, Taiwan
ieliao@nchu.edu.tw

***Abstract*—As more and more Internet of Things (IoT) devices are connected to the cloud services, fog computing has emerged as a new computing model to provide analytic and intelligent services to the end users for fast response time. Among the application layer protocols used for IoT, MQTT and CoAP are two well-known lightweight protocols that run on top of TCP and UDP, respectively. MQTT supports publish/subscribe pattern, while CoAP adopts request/response pattern. In IoT applications, publish/subscribe model was thought as a better model due to lower network bandwidth and less message processing which in turn extending the lifetime of battery-run devices.**

In this paper, we implement the publish/subscribe pattern in fog computing to preserve the privacy of customer energy usage data in smart grid environment using CoAP as the underlying application layer protocol. The performance of CoAP vs. MQTT in this model is also reported.

Keywords-- Fog Computing, IoT Application Layer Protocol, MQTT, CoAP, Smart Grid

I. INTRODUCTION

As more and more Internet of Things (IoT) devices are connected to the cloud services, the computation and communication latency between user devices and cloud data centers may not meet the requirement of some IoT applications that need to analyze and act on data in less a second. In recent years, fog computing or edge computing emerged as an alternative computing model for solving this problem, especially for the IoT applications in smart grids, smart transportation, and smart healthcare [1, 2, 3, 4].

IoT services including smart grid and traffic monitoring are usually based on the design of publish/subscribe pattern. In the case of smart grid, smart meters play the role of publishers; the Meter Data Management System (MDMS) plays the role of broker; the power generators, the aggregators, and the energy users act as subscribers.

There are three widely used application layer protocols for IoT applications, which are Extensible Messaging and Presence Protocol (XMPP), Message Queuing Telemetry Transport (MQTT), and Constrained Application Protocol (CoAP). XMPP was standardized by the Internet Engineering Task Force (IETF) in 2004 with subsequent Request for Comments (RFCs) in recent years [5]. XMPP runs over TCP. It provides publish/subscribe as well as request/response messaging systems. MQTT was designed to target lightweight machine to machine (M2M) communication, and it became an ISO standard (ISO/IEC 20922:2016) [6]. MQTT provides an asynchronous publish/subscribe protocol that runs on top of TCP/IP. CoAP was designed by IETF using a subset of the HTTP methods to target constrained-recourse devices. It runs over UDP and uses a synchronous request/response model. On July 4, 2017, IETF released the draft of Publish-Subscribe Broker for the Constrained Application Protocol (CoAP) [7]. The main purpose of this draft is to extend the capabilities of CoAP for supporting nodes with long breaks in connectivity and/or up-time.

In the application scenario of smart grid, the privacy of the customer energy usage data should be protected from leakage along the way from smart meters to concentrators to MDMS and to the stakeholders of power grid. Our research team proposed a model for aggregation and filtering on encrypted XML streams in fog computing to preserve the privacy of customer energy usage data [8]. In this paper, we present the publish/subscribe pattern used in the model using CoAP as the underlying application layer protocol. The performance of CoAP vs. MQTT in the model is also reported.

The remainder of this paper is organized as follows. Section 2 discusses related work. The proposed method is described in Section 3. Section 4 shows the experimental results. Section 5 gives the conclusion of this paper.

II. RELEATED WORK

A. Fog Computing

Fog computing is a computing concept proposed by Cisco [2]. Its idea comes from the edge computing. Because the computing nodes of edge servers are close to end devices, it

utilizes the computing resources of the edge nodes to provide immediate services. CARDAP [9] and Rainbow [10] platforms were designed based on the concept of fog computing for smart city services.

B. Communication Protocol and Publish Subscribe Pattern

Because IoT devices have the features of less computing resources and low power consumption, two light-weight application layer protocols MQTT [6] and CoAP [7, 11] are widely used in IoT applications. MQTT is introduced by IBM. Its design is based on topic oriented publish/subscribe pattern. It uses the TCP/IP to support reliable network communication. CoAP was introduced by IETF. Its design is based on request/response pattern by using the methods GET, POST, PUT, and DELETE in Constrained RESTful Environments (CoRE) to support end-to-end points communication. Unlike MQTT, CoAP runs on top of UDP. To achieve reliability of message transmission, it implements CON(confirmable), NON (Non-confirmable), ACK and RST signals to ensure the data being successfully delivered.

Several qualitative or quantitative comparisons of MQTT and CoAP have been reported in literatures [12, 13, 14]. Mijovic et al. [14] showed that CoAP achieves the highest protocol efficiency and the lowest average round trip time. However, the original request/response model of CoAP is a deficiency for its application to IoT environment. That's why the Publish-Subscribe Broker for CoAP is proposed by IETF to remedy this problem.

C. Security and Privacy-Preserving Schemes

Privacy and security issues are major concerns of the users of IoT applications. TLS (Transport Layer Security)[15] and DTLS (Datagram Transport Layer Security)[16] can be used in TCP and UDP, respectively, to provide confidentiality and integrity in the processes of data transmission. To encrypt the data, classical algorithms such as AES and RSA are commonly used. In fog computing environment, fog nodes may need to aggregate energy usage data before sending aggregated data to stakeholders of smart grid. Therefore, the encrypted energy usage data should be decrypted, and then computations can be performed. Because the fog nodes, for example, the concentrators of smart grid, are located in the network edge near users, the security protection of fog nodes may not be at the same level as that of the data centers. In this case, it will give the chance to hackers to obtain sensitive private data during the processing of decrypted energy usage data in the fog nodes. To avoid this problem, homomorphic encryptions enable mathematical operations to operate on encrypted data. Garcia et al. [17] proposed an approach which integrates the Paillier system and secret sharing technique to aggregate the energy consumption. However, the Paillier-based systems do not support multiplicative operations on encrypted data. Gentry [18] proposed a fully homomorphic encryption. But, its implementation still had poor performance. Selection of partial or fully homomorphic cryptographic systems depends on the applications used. To achieve the performance on additive operations, we selected the Paillier algorithm [19] as the basis of our encryption system.

III. Proposed Model

In this section, we describe a fog computing model for an IoT application in smart grid. In this model, smart meters are the sensor nodes; concentrators are the fog nodes; the meter data management system (MDMS) in the cloud is the management server/administrator.

A. System Archiecture

The system architecture of the proposed model is shown in Figure 1. In addition to the publishers, brokers, and subscribers in the publish/subscribe pattern, we add a management server or administrator to serve as the MDMS in smart grid. In fog computing, these four roles correspond to sensor nodes, fog nodes, service user nodes representing the stakeholders of smart grid, and the cloud servers. In an application scenario, subscribers such as power generators may need to know real-time power consumption in a region, then the subscribers submit their requests to the MDMS, which in turn updates the queries in the brokers. The brokers will filter and aggregate all of the encrypted energy usage data sent from sensors in that region without decrypting the data.

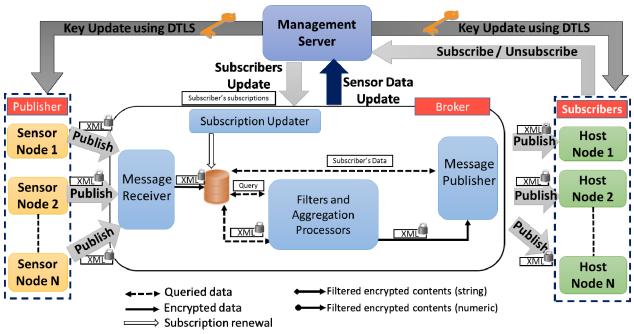

Figure 1. The system architecture

We assume that the publishers, brokers, and subscribers have registered in the management server in the cloud, and they have their own private keys which are transmitted by the management server over a DTLS protocol. To efficiently encrypt and decrypt the sensor data, which is represented in XML format, we use symmetric encryption algorithms AES and Paillier as our cryptosystems. Publishers and subscribers share the same secret keys of AES and Paillier. The overall workflow of the model is as follows:

1. Subscribers submit their individual subscriptions, for example, total power consumption in a region, to the administrator through the DTLS channel.
2. The administrator receives the subscriptions, and it transforms the subscription requests into the appropriate subscription rules for the broker. The subscription rules consist of XPath query patterns. Then, it updates the subscription rules to the broker.

3. The broker receives the updated rules and changes its settings of filters and aggregation processors.
4. The publishers publish their encrypted XML data to the broker by XML streams. In the smart grid application, each sensor node publishes its energy usage data containing householder, device ID, time, and readings.
5. When the broker receives the encrypted XML data using its message receiver, it filter the messages that match the queries in the broker and do the necessary aggregations on the filtered encrypted data. Then, it uses the message publisher to publish the aggregated encrypted XML data to the subscribers.
6. After receiving the aggregated encrypted data from the broker, the subscribers decrypt the data and retrieve the aggregated data that represent the total power consumption in a region..

B. *Publish/Subscribe Pattern*

In CoAP, all interfaces of each nodes are specified by Universal Resource Identifiers (URI). The URIs support RESTful methods for data transmission, and these methods are GET, POST, and PUT. Figure 2 shows the diagram of the methods used for the four roles in the proposed system. In this publish/subscribe model of CoAP, the published or subscribed resources are represented by different topics. For instance, a topic with URI coap://192.168.1.1:5683/Energy_Consumption represents the resources of encrypted consumption from a smart meter. The subscribers use GET and POST methods to subscribe these topics and submit the requests to the administrator. The administrator uses PUT method to update the queries for the broker and the keys for subscribers and publishers. The updated information includes the deployed secret keys and parameters of selected encryption/decryption algorithms to the subscribers and the publishers and the subscription rules to the brokers. The publishers use PUT method to publish the encrypted XML data to the topic URIs of the broker. In the draft of CoAP publish/subscribe pattern released by IETF [7], the broker notifies the subscribers of the update of the topics, and the subscribers use GET method to retrieve the resources of the topics from the broker. In the case of real-time services, it may generate unnecessary communication costs. Instead, we implement the broker such that it can actively notify and send the subscribed information to the subscribers. The broker uses PUT method to publish the matched and aggregated encrypted XML data to the URIs for the subscribers subscribing the topics.

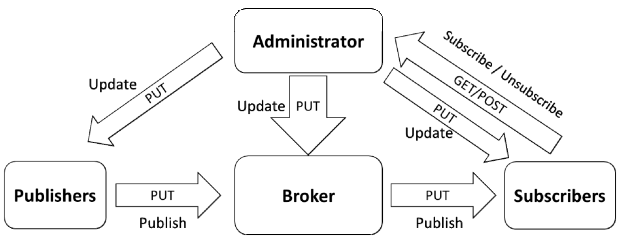

Figure 2. The RESTful methods in the publish/subscribe pattern

C. *Publisher*

Publishers are the IoT devices such as smart meters that publish their data in regular basis. The published data are represented in XML format. For example, the energy consumption record for the householder Andy is shown in Figure 3. The XML data consist of several elements including EnergyConsumption, Householder, Device, Id, Time, Record, Reading, and Unit. It describes Andy's Device1 consumes 20W at the recording time. Then, the publisher encrypts the XML data before transmission. The contents of XML elements are encrypted by different encryption algorithms. The string-type contents are encrypted using AES with 256-bit keys. The numeric-type data are encrypted using Paillier. In order to make brokers easily filter the needed data according to XPath query patterns without any decryption operations, publishers do not encrypt the XML elements. Instead, they use one-way hash algorithm like MD5 to generate corresponding cipher texts of XML elements. Using the proposed encryption methods, the encrypted XML contents of Figure 3 are shown in Figure 4.

Afterward, publishers use PUT method to put the encrypted data to the broker.

```
<?xml version="1.0" encoding="UTF-8"?>
- <EnergyConsumption>
    <Householder>Andy</Householder>
  - <Device>
      <Id>Device1</Id>
    </Device>
    <Time>2017-04-30 00:00:00</Time>
  - <Record>
      <Reading>20</Reading>
      <Unit>Watt</Unit>
    </Record>
</EnergyConsumption>
```

Figure 3. The example of the XML data

Figure 4. The encrypted XML data

D. *Broker*

In the broker, it maintains a small database to store the subscription information from the administrator. The subscription information consists of subscribers' CoAP URIs, the corresponding XPath query patterns, and the data types for the query patterns.

When the broker receives the published XML data from the publishers, it will do the following actions:

1. Filter the necessary contents in the XML data according to the XPath query patterns from the subscription rules stored in the database.
2. Classify the filtered contents that matched the subscription rules into string-type and numeric-type.
3. Perform the aggregation computation using the Paillier – based algorithm [19] for the numeric values.
4. The Message publisher in the broker assembles the string-type contents and the aggregated numeric-type contents into a message and then uses PUT method to publish the message to the CoAP URIs of the subscribers.

To efficiently filter the incoming XML data streams from many of IoT devices, the broker node is designed to run a number of concurrent XPath filters. The number of XPath filters is configurable. The experimental results in the next section also show the results for different numbers of filters.

The broker also receives the update requests of the subscription information from the administrator. These requests are categorized into three classes: insertion, deletion, and modification, which corresponds to new subscriptions, un-subscriptions, and changes of subscription contents, respectively.

E. Subscriber

Through a DTLS channel between a subscriber and the administrator, the subscriber does the following actions to complete a subscription request on a topic.

1. The subscriber submits a request through GET method to the URI of the topic in the administrator.
2. After receiving the acknowledgement message representing successful subscription from the administrator, it prepares the URI for receiving the subscribed data from the broker. The host name of the available broker is given in the payload of the acknowledgement message.
3. The subscriber uses POST method to submit its prepared URIs to the administrator.
4. If the subscriber receives the response code 2.05 (Content) from the administrator, it actually completes the subscription operation.

F. Administrator

The administrator or message server stores the URIs of the topics for the subscribers. If it receives a subscription request from a subscriber through GET method, it will complete the following operations:

1. It authenticates the subscriber using DTLS and examines the request for the topic.
2. If the request is valid, it returns a response code 2.05 (Content) and the corresponding payload to the subscriber. The payload includes the host name of the broker. Then it waits for the PUT request representing the prepared URIs from the subscriber.
3. If the request is not valid, it returns the response code 4.01 (Unauthorized).
4. If it receives the PUT request, it returns a response code 2.05 (Content) to the subscriber. Then, it generates the rules of the subscription and updates the rules including the XPath patterns, subscribers' URI in the database of the broker through PUT method. If the update is successfully performed, the subscription process is completed.

The un-subscription operations are can be done in the same way.

IV. EXPERIMENTAL RESULTS

In this section, we describe the system environment for implementing the publish/subscribe pattern based on CoAP. Two versions of hardware platforms are used for the broker. One is a PC with AMD Athlon(tm) II X4 640 3.0G 64-bit processor and 8GB memory. Another one is a Rasperberry Pi 3. We prepared a PC to simulate the publishers and the subscribers. The number of simulated publishers varies from 100, 200, and to 300. All of the software components are implemented using Java 1.8. The CoAP implementation is based on Eclipse Californium [20].

The performance of the implemented CoAP publish/subscribe pattern will be compared with that of MQTT. We used the same hardware environment for MQTT, and its implementation is based on Eclipse Paho project [21]. From the viewpoint of the architectural design, the main bottleneck of the overall system is on the broker. Therefore, we only measure the execution time of the broker for CoAP and MQTT. We prepared three encrypted XML files with sizes of 2KB, 4KB, and 6KB for the publishers. To evaluate the performance of the broker, we measure its execution time starting from receiving the first encrypted XML file until generating the result message using message publisher. To fairly evaluate the performance, the reported execution time is the average of 10 executions by the broker. The number of concurrent XPath filters for the proposed model varies from 5, 10, and to 15. The key updates for CoAP and MQTT are performed using DTLS and TLS, respectively.

The first experiment is performed using 100, 200 and 300 publishers with the same encrypted XML file of size 6KB as the input. The experimental results using PC or Rasperberry Pi as the broker are shown in Table 1, Table 2, and

Table 3 for different number of concurrent XPath filters. We can see that the performance of CoAP obviously outperforms that of MQTT. The overall execution time on Raspberry Pi takes much longer than that of PC because of the limited hardware resources in Raspberry Pi. In the PC environment, the execution time is reduced when the number of XPath filters is changed from 5 to 10. However, the execution time is worse in case of increasing XPath filters from 10 to 15. That's because the limitation of available hardware resources. The same situation also happened on the Raspberry Pi platform when the number of concurrent filters is increased from 5 to 10.

Table 1. The execution time using 5 concurrent XPath filters (in micro seconds)

Number of Sensor Nodes	PC		Raspberry Pi	
	CoAP	MQTT	CoAP	MQTT
100	1686.4	1890.8	5102.6	8833.2
200	3716.6	3901.6	11512.8	20058
300	6346.8	5790.4	17544	30604

Table 2. The execution time using 10 concurrent XPath filters (in micro seconds)

Number of Sensor Nodes	PC		Raspberry Pi	
	CoAP	MQTT	CoAP	MQTT
100	1627.6	1814.4	5191.2	8884.6
200	3665.4	3752.2	11522.4	20106.4
300	6245	5638.8	17890.6	30995.4

Table 3. The execution time using 15 concurrent XPath filters (in micro seconds)

Number of Sensor Nodes	PC		Raspberry Pi	
	CoAP	MQTT	CoAP	MQTT
100	1668.8	1853.2	5196.2	8898.8
200	3796	3822.6	11728.8	20210.8
300	6320	5690.6	17893.8	31192.4

We also compared the performance of CoAP and MQTT with 5 filters in the broker using different sizes of encrypted XML file in the publishers. As shown in Table 4, the execution time increases when the size of the encrypted data is increased from 2KB to 6 KB. In any case, the performance of CoAP is much better than that of MQTT.

Table 4. The execution time using different sizes of encrypted XML files

Data Size	The number of publishers using CoAP		The number of publishers using MQTT		
	100	200	300	100	200	300
2KB	4438.2	10072.2	15180.4	8623	19679.4	29324.8
4KB	4746.2	10689.8	15948.8	8793	19954.2	30501.2
6KB	5102.6	11512.8	17544	8859.6	20058	30604

V. CONCLUSIONS

The IoT applications are blooming due to the hardware and software development of the connected world. It is interesting to see how two competing protocols CoAP and MQTT are adopted in real applications in the future. However, MQTT uses publish/subscribe pattern which might be more suitable in IoT applications, whereas CoAP adopts request/response pattern. Another technology shift is that the fog computing architecture has been seen as the next generation computing model for providing fast analytic and intelligent services to the users.

In this paper, we implement the publish/subscribe pattern in fog computing environment for a smart grid application that preserves the privacy of customer energy usage data using CoAP as the underlying application layer protocol. The performance of CoAP is also evaluated and compared with that of MQTT. The experimental results show that CoAP outperforms MQTT in all cases of varying the number of concurrent filters in the broker, the number of sensor nodes, and different sizes of data published by publishers. Therefore, the findings of this paper can be used as a reference for future implementation of IoT applications.

ACKNOWLEDGEMENT

This research was partially supported by the Ministry of Science and Technology, Taiwan, under the grant number MOST 105-2221-E-005-070 and by the Taiwan Information Security Center at NCHU (TWISC@NCHU), Ministry of Science and Technology, Taiwan, under the grant numbers MOST 105-2218-E-001-001 and MOST 106-3114-E-005-001.

REFERENCES

[1] P. Zhang, F. Li, and N. Bhatt, "Next-Generation Monitoring, Analysis, and Control for the Future Smart Control Center," *IEEE Transactions on Smart Grid,* vol. 1, no. 2, July 2010, pp. 186-192.

[2] CISCO, "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are," Available: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in Proceedings of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, August 17, 2012, pp. 13-16.

[4] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog Computing: Principles, Architectures, and Applications," in *Internet of Things*: Morgan Kaufmann, 2016.

[5] Internet Engineering Task Force , "RFC-6120 Extensible Messaging and Presence Protocol (XMPP): Core," March 2011,Available: https://tools.ietf.org/html/rfc6120

[6] ISO, "ISO/IEC 20922:2016 Information technology -- Message Queuing Telemetry Transport (MQTT) v3.1.1," 2016, Available: https://www.iso.org/standard/69466.html

[7] Internet Engineering Task Force , "Publish-Subscribe Broker for the Constrained Application Protocol," July 2017,Available:

https://tools.ietf.org/html/draft-ietf-core-coap-pubsub-02

[8] J.-Y. Huang, W.-C. Hong, P.-S. Tsai, and I.-E. Liao, "A model for aggregation and filtering on encrypted XML streams in fog computing," International Journal of Distributed Sensor Networks, Vol. 13(5), 2017, pp. 1-14.

[9] P. P. Jayaraman, J. B. Gomes, H. L. Nguyen, Z. S. Abdallah, S. Krishnaswamy, and A. Zaslavsky, "CARDAP: A Scalable Energy-Efficient Context Aware Distributed Mobile Data Analytics Platform for the Fog," in Proceedings of 18th East European Conference on the Advances in Databases and Information Systems, Ohrid, Macedonia, September 7-10, 2014, pp. 192-206.

[10] A. Giordano, G. Spezzano, and A. Vinci, "Smart Agents and Fog Computing for Smart City Applications," in Proceedings of International Conference on Smart Cities, Malaga, Spain, September 12-15, 2016, pp. 137-146.

[11] M. Castro, A. J. Jara, and A. F. Skarmeta, "Enabling End-to-End CoAP-based Communications for the Web of Things," Journal of Network and Computer Applications , Vol. 59, No. C, 2016.

[12] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, C. K.-Y. Tan, "Performance Evaluation of MQTT and CoAP via a Common Middleware," in Proceedings of IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), April 21-24, 2014, pp. 1-6.

[13] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A Survey on Application Layer Protocols for the Internet of Things," Transaction on IoT and Cloud Computing, Vol. 1, No. 1, January 2015, pp. 1-10.

[14] Mijovic, E. Shehu, and C. Buratti, "Comparing Application Layer Protocols for the Internet of Things via Experimentation," Research and Technologies for Society and Industry Leveraging a better tomorrow(RTSI), Bologna, Italy, Sept. 7-9, 2016.

[15] Internet Engineering Task Force, " Transport Layer Security Version 1.2," August 2008, Available: https://tools.ietf.org/pdf/rfc5246.pdf

[16] Internet Engineering Task Force, "Datagram Transport Layer Security Version 1.2," January 2012, Available: https://tools.ietf.org/pdf/rfc6347.pdf

[17] F. D. Garcia and B. Jacobs, "Privacy-Friendly Energy-Metering via Homomorphic Encryption," in Proceedings of the 6th International Conference on Security and Trust Management, Athens, Greece, September 23-24, 2010, pp. 226-238.

[18] C. Gentry, *A fully homomorphic encryption scheme*. 2009.

[19] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, pp. 223-238.

[20] Eclipse. *Eclipse Californium*. Available: http://www.eclipse.org/californium/#top

[21] Eclipse. *Eclipse Paho*. Available: https://eclipse.org/paho/