Security in MQTT and CoAP Protocols of IOT's Application Layer

Anup Burange, Harshal Misalkar(✉), and Umesh Nikam(✉)

Amravati, India
{awburange, hdmisalkar, uvnikam}@mitra.ac.in

Abstract. The Internet of Things (IoT) is a framework of interconnected computing devices mechanical and digital machines, internationally identifiable physical objects (or things) or people that are have unique identity and the ability to transfer data over a network without human-to-human or human-to-computer interaction., their combination with the Internet, and their representation in the digital world. The accessibility and availability of cheap components of IoT devices enables a extensive range of applications and provide smart environments. These devices perform actuating and sensing tasks and identified through unique addresses. The IoT devices are connected to the Internet and expected to use the Constrained Application Protocol (CoAP) at the application layer as a main web transfer protocol. Message Queuing Telemetry Transport (MQTT) does not enforce the use of a particular security approach for its applications, but instead leaves that to the application designer. Therefore, IoT solutions can be based on application context and specific security requirements. MQTT is a Client Server publish/subscribe messaging transport protocol. It is lightweight, open, uncomplicated, and designed to make implementation more easier. These characteristics of MQTT make it perfect for use in most of the situations, including communication in Machine to Machine (M2M) and Internet of Things (IoT). In IOT there is major use of Wireless Sensor Networks (WSN) which connects virtual world to physical world. In this paper focus is given to application layer of IOT. In application layer two important protocols are MQTT and CoAP. Security mechanism is proposed in the paper for these protocols.

Keywords: MQTT · CoAP · IOT

1 Introduction

The IoT is built on three main pillars related to the capability of objects which must have communication capability, computational capability and may have interaction capability:

(i) Communication capability: Objects in IoT must have a minimal set of communication capability. What we mean by this is not only a communication channel, but also everything related to it, in order to make an efficient communication, such as, an address, identifier, and name. The objects may have all these features or some of them [8].

© Springer Nature Singapore Pte Ltd. 2019
S. Verma et al. (Eds.): CNC 2018, CCIS 839, pp. 273–285, 2019.
https://doi.org/10.1007/978-981-13-2372-0_24

(ii) Computational capability: Objects must have some basic or complex computational capability, in order to process data and networks configurations. For instance, receive commands over the communications channel, manage network tasks, save the status from a sensor, activate an effector.
(iii) Interaction capability: The IoT technology may have an interaction capability in terms of sensing and actuating. This can be done either by, sensors and/or actuators. Sensors are things which sense or detecting the real world environment (e.g., light, humidity, temperature, movement, voice, etc.). Effectors are things that change or effect the real world such as, switches that allow you to trigger or to turn on/off anything that can change the real word such as motors, beepers, cameras, etc [1].

Table 1. IoT stack with standaridized security solutions.

IoT layer	IoT protocol	Security protocol
Application	CoAP	User-defined
Transport	UDP	DTLS
Network	IPv6, RPL	IPsec, RPL security
6LoWPAN	6LoWPAN	None
Data-link	IEEE 802.15.4	802.15.4 security

2 IOT Architecture

Many architecture models have been proposed from different organizations and researchers. Figure 1 provides the main and general model which consists of three-layer architecture perception layer, network layer, and application layer. The definitions of these layers are defined below:

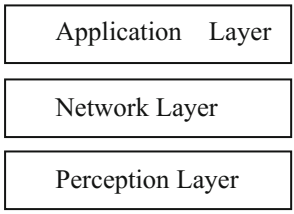


Fig. 1. IOT architecture

1. Perception layer: The perception layer can also be called physical layer. This layer consists of physical objects or devices such as sensors, RFID system, meters, GPS system. This layer basically collect identification or information depends on the type of the sensor or the physical device.
2. Network layer: The network layer is also known as transmission layer. This layer is responsible for interconnection and communication functions. The transmission can

be done through wired or wireless communication technologies such as Wi-Fi, Bluetooth, 3G, ZigBee. Many transmission and security protocols can be deployed in this layer such as: IPV4, IPV6, DTLS, IPsec. This layer should securely transfer the data from the physical layer to application layer.

3. Application layer: The application layer provides and manages application services for users needs. Many application protocols can be deployed in this layer such as CoAP, and HTTP depends on the type of application and the IoT devices [7].

The recent challenge in consideration with security of IOT devices is to fix security bugs & its security updation. In addition to new protocols that are designed specifically for the Internet of Things such as Message Queuing Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP), security mechanisms should need to be developed or upgrade [2].

3 MQTT (Message Queueing and Transport Protocol)

For IoT devices and applications, MQTT is the most recognized messaging protocol and it is the base of many active groups or industries in the IoT field. Lightweight, easy-to-use message protocols are being provided by MQTT as IOT solutions. MQTT involves some safety mechanisms in addition to common implementations such as SSL/TLS for transport protection [5]. For applications MQTT does not implement the use of a specific security approach, but in its place handover that task to the application designer, because of this IoT solutions are based on application structure and definite security necessities. MQTT uses transport layer security (TLS), for most of the deployments where data is encrypted and its reliability is validated. To control admittance most implementations of MQTT also use permission features in the MQTT server.

3.1 Architecture

Every sensor in a client/server model of MQTT is known as client that connects to a broker, that broker act as a server. Connection is established using TCP/IP. As MQTT is message oriented protocol, each message is having discrete amount of data, transparent to the broker. The broker is a server which can be installed on any machine. MQTT runs on TCP/IP layer therefore it is connection oriented protocol, every client must establish connection with the broker before starting communication. Every message is available to an address, identified as a topic. Clients may subscribe to several topics. Clients can subscribe to various topics. Each client subscribed to a topic gets every message published to the topic. Consider a simple network with three nodes that release TCP connections with the broker. Nodes Y and Z are subscribe to the topic humidity (Fig. 2).

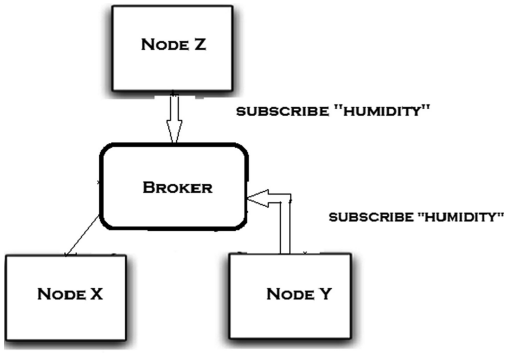


Fig. 2. Architecture of MQTT

Afterward Node X displays a value of 30% for topic humidity, then the broker transmits the message to all the subscribed notes.

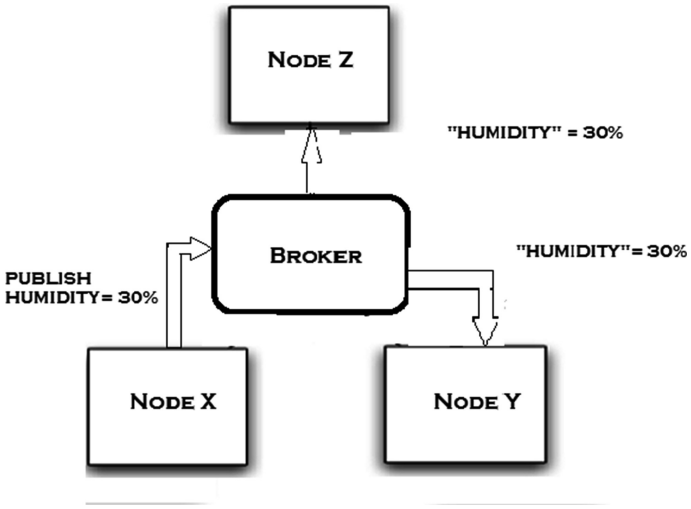


Fig. 3. Architecture of MQTT

The publisher-subscriber model on which MQTT works allows MQTT clients to communicate one-to-one, one-to-many and many-to-one.

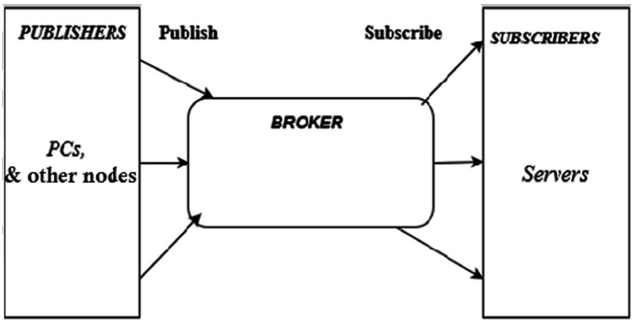


Fig. 4. Publisher-subscriber model of MQTT

3.2 MQTT Vulnerabilities

The MQTT protocol carries a number of potential vulnerabilities. For example, open ports can be used to launch denial-of-service (DoS) attacks as well as buffer overflow attacks across networks and devices [6]. Depending on the number of IoT devices connected and use cases supported, the complexity of "topic" structure can grow significantly and cause scalability issues. MQTT message payloads are encoded in binary, and corresponding application/device types must be able to interoperate. Another problem area is with MQTT message usernames and passwords, which are sent in clear text. Transport encryption with SSL and TLS can protect data when implemented correctly. To protect against threats, sensitive data including user IDs, passwords, and any other types of credentials should always be encrypted. To secure MQTT protocol we should consider the security of client, broker, operating system.

3.3 MQTT Security

By its nature, MQTT is a plain protocol. All the information exchanged is in plain-text format. In other words, anyone could access to this message and read the payload. There are several use cases where we want to keep information private and guarantee that it cannot be read or modified during the transmitting process. In this case, there are several approaches we can use to face the MQTT security problem:

1. Create a VPN between the clients and the server.
2. Use MQTT over SSL/TSL to encrypt and secure the information between the MQTT clients and MQTT broker.

Our attention is, on how to create an **MQTT over SSL**. To make MQTT a secure protocol, we have to follow these steps:

- Create a private key (CA Key).
- Generate a certificate using the private key (CA cert).
- Create a certificate for Mosquitto MQTT server with the key.

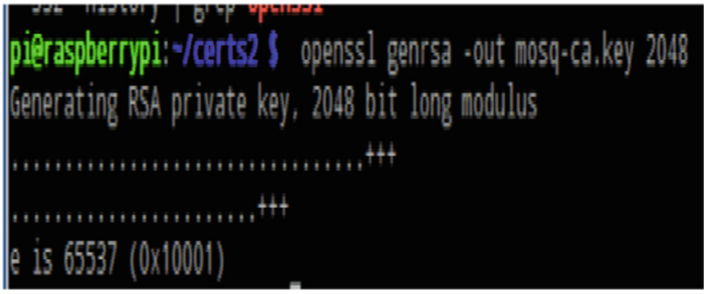
The final step is configuring Mosquitto MQTT so that it uses these certificates.

3.4 Securing MQTT Server

The first step in this process is creating a private key. Connect to the Raspberry Pi using ssh or a remote desktop and open a command terminal. Before starting, it is important to ensure OpenSSL is installed on Raspberry Pi.

[openssl genrsa -out mosq-ca.key 2048]

Using this command, we are creating a 2048-bit key called mosq-ca.key. The result is shown in the picture below:



The next step is creating an X509 certificate that uses the private key generated in the previous step. Open the terminal again and, in the same directory in which private key is stored, write:

[openssl req -new -x509 -days365 -key mosq-ca.key -out mosq-ca.crt]

In this step, we need to provide different information before creating the certificate.

- Country Name
- State or Provenance Name
- Locality Name
- Organization Name
- Organizational Unit Name
- Common Name
- Email Address.

3.5 Creating the MQTT Server Certificate

Once the private key and the certificate are ready, we can create the MQTT server certificate and private key:

$$[\text{openssl genrsa -out mosq-serv.key 2048}] \quad (1)$$

During this step, we need to create a CSR (Certificate Signing Request). This certificate should be sent to the Certification authority that, after verifying the author identity, returns a certificate. We will use a self-signed certificate:

$$[\text{openssl req -new -key mosq-serv.key -out mosq-serv.csr}] \quad (2)$$

we have used the private key generated in the step before. Finally, we can create the certificate to use in our MQTT Mosquitto Server:

$$[\text{openssl x509 -req -in mosq-serv.csr -CA mosq-ca.crt -CAkey mosq-ca.key -}$$

$$\textit{CAcreateserial} - \textit{out}\ \text{mosq-serv.crt -days 365 -sha256}] \quad (3)$$

All done! We have completed the steps necessary to secure our MQTT server.

$$[\text{openssl x509 -in mosq-serv.crt -noout -textjavascript:void(0)}] \quad (4)$$

4 CoAP (Constrained Application Protocol)

CoAP runs over UDP, not TCP. Clients and servers communicate through connectionless datagrams. Retries and reordering are implemented in the application stack. Removing the need for TCP may allow full IP networking in small microcontrollers. CoAP allows UDP broadcast and multicast to be used for addressing. CoAP follows a client/server model. Clients make requests to servers, servers send back responses. Clients may GET, PUT, POST and DELETE resources. CoAP employs a client-server model and request/response message pattern, where client devices send information requests directly to server devices, which then respond. Support for an observer message pattern enables clients to receive an update whenever a requested state changes, for example a valve opening or closing, while confirmed message delivery provides some level of assurance under the connectionless UDP transport.

The Constrained Application Protocol (CoAP) is a web transfer protocol at the application layer intended to be used with constrained devices The Internet Engineering Task Force (IETF) working group has designed this protocol to be used for M2M applications, IoT objects and suitable for constrained devices that have limited amount of ROM and RAM [9]. One design goal of CoAP is limiting the need for fragmentation by using small message overhead. Moreover, this protocol suitable for constrained networks such as 6LoWPAN which supports the fragmentation of IPv6 packets into small frames. CoAP provides an interaction model similar to the client/server of HTTP.

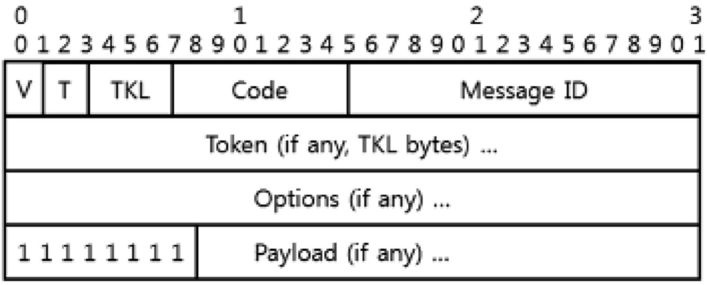


Fig. 5. CoAP message format

CoAP Architecture as shown it extends normal HTTP clients to clients having resource constraints. These clients are known as CoAP clients. Proxy device bridges gap between constrained environment and typical internet environment based on HTTP protocols. Same server takes care of both HTTP and CoAP protocol messages [4].

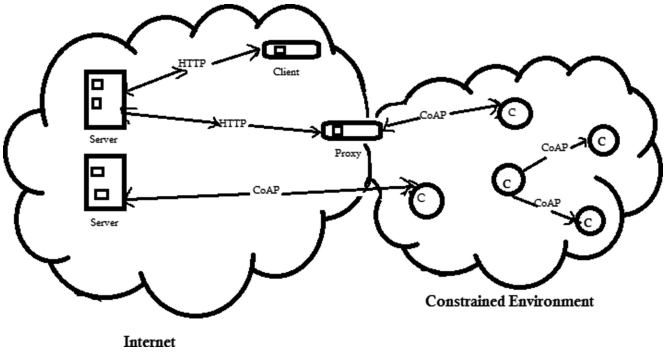


Fig. 6. CoAP architecture

4.1 DTLS for CoAP Security

DTLS is used more than CoAP to provide continuous security. Just as TLS being used for securing HTTP over TCP, Datagram TLS (DTLS) is used for securing CoAP over UDP. DTLS is implemented between transport layer and application layer as in Fig. 7. As DTLS operated on top of the UDP protocol, the complexity of its implementation increased which requires mechanisms to provide reliability. To design a DTLS version

that can be used in constrained environments, it is important to minimize the code size, and the number of messages exchanged to get an optimized handshake protocol.

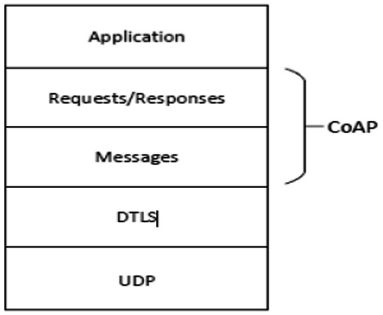


Fig. 7. Abstract Layering of DTLS-Secured CoAP

Figure 8 shows how the DTLS handshake works using CoAP, providing communication reliability by CON and ACK messages using CoAP block-wise transfer which contain DTLS handshake messages as a payload. When the DTLS handshake session has finished, the client can initiate the first CoAP request.

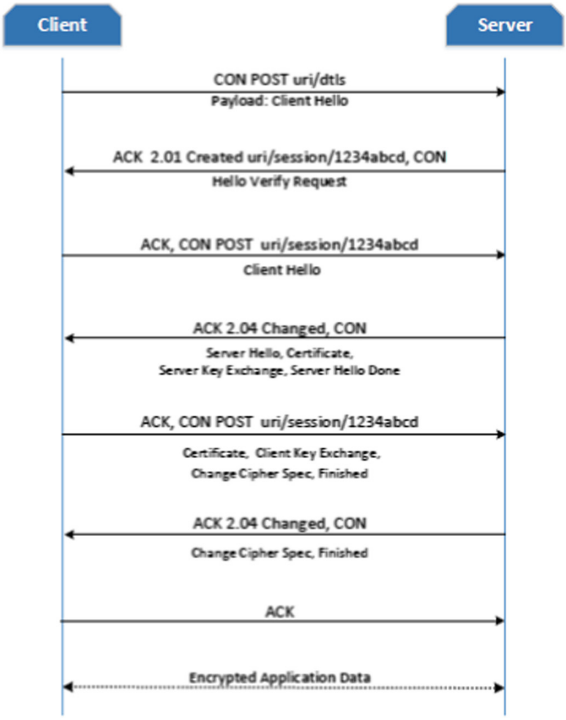


Fig. 8. CoAP protocol layers

4.2 DTLS over CoAP (CoAPs)

DTLS protocol can be integrated with CoAP to provide end-to-end security. In this implementation, we used the open source TinyDTLS and CoAP libraries. Tiny DTLS supports the cipher suite based on Pre-shared keys (PSK) with the Advanced Encryption Standard (AES): TLS PSK WITH AES 128 CCM 8. We implement two nodes a CoAP server and CoAP client, with an integration of Tiny DTLS for both server and client. We observe the output and compare the simulation result with the previous CoAP simulation.

The general interactions of data between DTLS and CoAP in both forward and reverse directions are illustrated in Fig. 9(a) and (b) respectively. In the forward direction, CoAP packets are sent to DTLS module to add the security functionality [3].

There are two interfaces in this operation: DTLS receives normal data packets from CoAP and then sends encrypted data to CoAP. Afterward, the encrypted packets are sent across to UDP as shown in Figure (a).

In the reverse direction, the secured packets received from UDP are sent across to DTLS for decryption then sending it back to CoAP as shown in Figure (b).

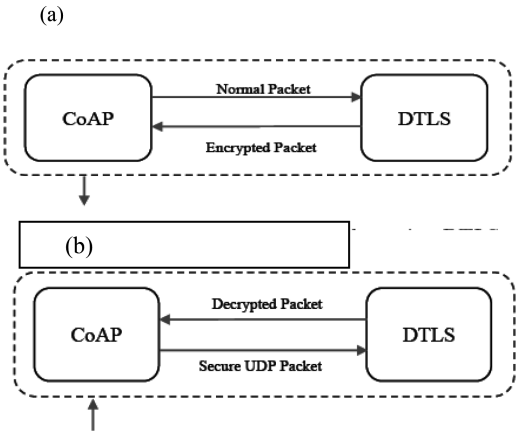


Fig. 9. (a) Encrypting a CoAP packet using DTLS (b) Sending a DTLS decrypted to CoAP

5 Experiment Results

IoT sensor devices have limited memory, CPU and power resources. In our experiment, we test the Constrained Application Protocol with and without security in constrained sensor nodes. In order to know the impact of deploying security mechanisms on constrained devices, we test and compare the memory footprint:

(a) Memory Footprint

The Internet Engineering Task Force group (IETF) classified constrained devices with consideration of code size and data size as shown in Table 2. Therefore, it is important

to know the code size and the data size for an application to measure the memory usage. The difficulty of providing a secure communication increase with limited resources. The classification of constrained devices is shown below:

Class 0 devices are very constrained nodes. They have limited memory and processing capabilities and may not have enough resources to communicate securely and directly to the Internet. Thus, they need a help of larger devices such as a proxy or gateway to participate in Internet communications.
Class 1 devices are quite constrained nodes. They cannot easily use full protocol stack such as HTTP or TLS. However, they are capable to use protocols that designed for constrained nodes such as CoAP over UDP. In our implementation, we use Wismote node which has 16 KB of RAM and 128 KB ROM and considered as a class 1 device.
Class 2 devices can support most protocols used in Internet communication.

Table 2. Classes of constrained devices

Name	Data size (e.g., RAM)	Code size (e.g., Flash)
Class 0	$\ll$10 KB	$\ll$100 KB
Class 1	$\sim$10 KB	$\sim$100 KB
Class 2	$\sim$50 KB	$\sim$250 KB

The memory footprint is provided by the MSP430-GCC compiler. We obtained the RAM and ROM by utilizing the MSP430-GCC size command of the firmware files. Table 3 shows the require memory size for both server and client with and without security implementation for Wismote sensor. Figures 10 and 11 show the impact of deploying security for CoAP. The difference of the RAM size between the two codes is about 30%. Whereas, the flash memory or ROM has increased by approximately 59%.

Table 3. Memory footprint

Node type	RAM [bytes]	ROM [bytes]
CoAP Client	8210	44831
CoAP Server	8200	50224
CoAPs Client	11396	86583
CoAPs Server	11274	89737

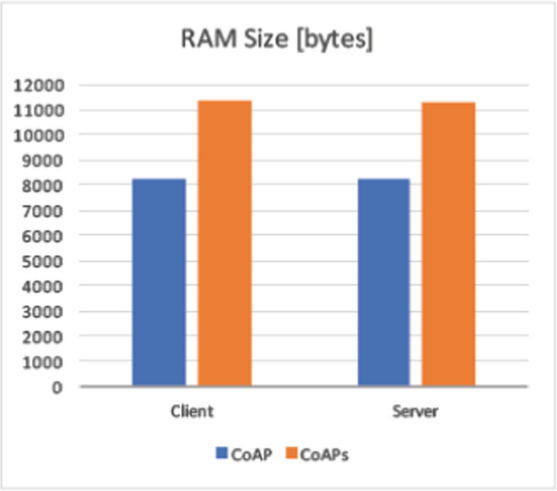


Fig. 10. RAM footprint

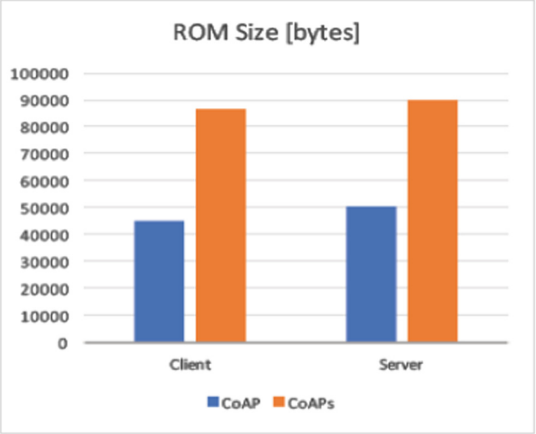


Fig. 11. ROM footprint

6 Conclusion

The Internet of Things is considered as one of the largest improvement to the existing technology nowadays. It is extremely important to have a secure IoT system in order to develop and improve this technology to be used in a large scale. In this paper, we address the fundamentals of the Internet of things and the key features and requirements. Followed by the the Constrained Application Protocol (CoAP) as it will be a significant part of IoT. We present an overview of the Datagram Transport Layer Protocol (DTLS) which is one way of providing an end-to-end security for IoT

applications. We also addressed security techniques for securing MQTT protocol, which is also one of the important protocol of IOT.

References

1. Asim, M.: A survey on application layer protocols for Internet of Things (IoT). Int. J. Adv. Res. Comput. Sci. **8**(3), 996–1000 (2017). ISSN 0976-5697
2. Kraijak, S., Tuwanut, P.: A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends. In: 11th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2015), pp. 1–6, September 2015. https://doi.org/10.1049/cp.2015.0714
3. Rahman, R.A., Shah, B.: Security analysis of IoT protocols: a focus in CoAP. In: 2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC), pp. 1–7. IEEE (2016)
4. Ugrenovic, D., Gardasevic, G.: CoAP protocol for web-based monitoringin IoT healthcare applications. In: 2015 23rd Telecommunications Forum Telfor(TELFOR), pp. 79–82, November 2015
5. Thangavel, D., Ma, X., Valera, A., Tan, H.-X., Tan, C.K.-Y.: Performance evaluation of MQTT and CoAP via a common middleware. In: IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing 2014, Singapore (2014). ISSNIP.2014.6827678
6. Chen, M., Wan, J., Gonzalez, S., Liao, X., Leung, V.C.M.: A survey of recent developments in home M2M networks. IEEE Commun. Surv. Tutor. **16**(1), 98–114 (2014). First Quarter
7. Wang, M., Zhang, G., Zhang, C., Zhang, J., Li, C.: An IoT-based appliance control system for smart homes. In: 2013 Fourth International Conference on Intelligent Control and Information Processing (ICICIP), pp. 744–747, 9–11 June 2013
8. Miorandi, D., Sicari, S., De Pellegrini, F., Chlamtac, I.: Internet of Things: vision, applications and research challenges. Ad Hoc Netw. **10**(7), 1497–1516 (2012)
9. Ishaq, I., Hoebeke, J., Moerman, I., Demeester, P.: Experimental evaluation of unicast and multicast CoAP group communication. Sensors **16**(7), 1–8 (2016). NCBI