

Toward a Security IoT Platform with High Rate Transmission and Low Energy Consumption

Lam Nguyen Tran Thanh[1(✉)], The Anh Nguyen[2], Hong Khanh Vo[2], Hoang Huong Luong[2], Khoi Nguyen Huynh Tuan[2], Anh Tuan Dao[2], and Ha Xuan Son[3]

[1] VNPT Information Technology Company, Ho Chi Minh city, Vietnam
[2] FPT University, Can Tho city, Vietnam
[3] University of Insubria, Varese, Italy

Abstract. The Internet of Things currently is one of the most interesting technology trends. Devices in the IoT network towards mobility and compact in size, so these have a rather weak hardware configuration. One of the essential concern is energy consumption. There are many lightweights, tailor-made protocols for limited processing power and low energy consumption, of which MQTT is the typical protocol. A light and simple protocol like MQTT, however, has many problems such as security risks, reliability in transmission and reception. The current MQTT protocol supports three types of quality-of-service (QoS). The user has to trade-off between the security/privacy of the packet and the system-wide performance (e.g., transmission rate, transmission bandwidth, and energy consumption). In this paper, we present an IoT Platform Proposal to improve the security issues of the MQTT protocol and optimise the communication speed, power consumption, and transmission bandwidth, but this still responds to reliability when transmitting. We also present the effectiveness of our approach by building a prototype system. Besides, we compare our proposal with other related work as well as provide the complete code solution is publicized to engage further reproducibility and improvement.

Keywords: Internet of Things · MQTT · Quality-of-Service (QoS) · Single Sign On · Kafka

1 Introduction

In recent years, the Internet of Thing (IoT) applications have grown and applied in most fields of our life such as smart city, healthcare, supply chains, industry, agriculture. According to estimates, by 2025, the whole world will have approximately 75.44 billion IoT connected devices [1]. Timothy Chou et al. [2] claimed that the IoT system architecture consists of five layers in order from low to high: Things, Connect, Collect, Learn, and Do.

The Things layer contains actuators that control physical devices or sensors to collect environmental parameters. The Connect layer connects devices with

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12949, pp. 647–662, 2021.
https://doi.org/10.1007/978-3-030-86653-2_47

applications and users. The Collect layer is responsible for aggregating the data returned by the Things layer devices. Finally, Learn layer is used to analyze data to give suggestions to the Do layer in response to received data. The Things and Connect layers can be considered as the two most important since they provide input data for the upper layers.

The Things layer usually consists of devices in the IoT which have limitations in network connectivity, power, and processing capabilities [3]. Therefore, how can we optimize the processing capacity and energy consumption of the devices while still having to meet some basic requirements on communication speed and the confidentiality of information for the whole system? This issue is determined by the protocols themselves within the Collect layer.

There are 5 popular protocols used for IoT platform, namely Hypertext Transfer Protocol (HTTP), Constrained Application Protocol (CoAP), Extensible Messaging and Presence Protocol (XMPP), Advanced Message Queuing Protocol (AMQP), and Message Queuing Telemetry Protocol (MQTT) [4]. For communications in limited networks (constrained networks), MQTT and CoAP are proposed to use [5]. Corak et al. found that the MQTT protocol has faster Packet creation time and Packet transmission time is twice as fast as with the CoAP protocol [6]. For developers of low bandwidth and memory devices, MQTT is also the most preferred protocol [7]. In addition, a comparison of the energy consumption level shows that the MQTT protocol consumes less energy than CoAP [8]. Therefore, we assess the energy consumption as well as the existing security risks of the MQTT protocol.

The MQTT protocol has three levels of QoS, ranging from 0 to 2[1] and these QoS levels are related to the level of confidence in the transmission of the packet (i.e., QoS has the lowest confidence level and QoS-2 has the confidence level highest reliability). Shinho et al. [9] demonstrate that the packet loss rate and the packet transmission rate of the QoS-0 level are the highest. Besides, Jevgenijus et al. [10] found that the energy consumption of the QoS-0 level is only about 50% of the QoS-2 level and the communication bandwidth of QoS-0 is also lower than that of QoS-2. Therefore, in this paper, we present a system design that takes advantage of the fast speed, low energy consumption and bandwidth while still meeting the reliability of the QoS-2 level.

The current MQTT protocol only provides identity, authentication and authorization for the security mechanism [11]. L. Lundgren et al. [12] indicate that data can be obtained by subscribing to any topic of the MQTT broker that is public on the internet. This is a serious limitation in terms of security. If the attacker subscribes to MQTT topics with the client ID, the victim will experience a denial of service (DOS) state and all information sent to the victim will be forwarded to the attacker [13].

MQTT supports authentication by username and password pair for authorization mechanism, but this authentication mechanism is not encrypted [11]. According to a survey by Zaidi et al. [14] about Shodan, the world's first IoT search engine for Internet-connected devices, points out that there are 67,000

[1] https://mqtt.org/.

MQTT servers on the public Internet with most of them without authentication. For authorization, access to a specific topic based on the access list (ACL). This mechanism must be defined in advance in the MQTT broker configuration file and must restart the MQTT broker service if the users want to apply the new access-list configuration. This is inconvenient and difficult to expand especially for systems with billions of devices [15].

Besides MQTT's security flaws, IoT systems are vulnerable to user behavior. Users often tend to ignore security issues especially privacy until the loss of critical data [16]. Subahi et al. [17] found that a significant proportion of IoT users were not fully aware of their behavior. Therefore, with IoT systems having billions of users and devices, it is difficult to manage the behavior of all users.

In this paper, we propose the model of IoT Platform using message queue, which is able to take advantage of the reliable transmission capability of MQTT QoS-2 but still satisfies the fast message transmission rate. Besides, this paper shows how to combine the MQTT and Oauth protocols to improve the Authentication and Authorization mechanism of the current MQTT protocol. We also offer a management model of users, things and channels information to prevent DOS attacks and attack the availability of systems using MQTT protocol. In addition, we offer a method of strict management of the communication process to ensure that users control their information-sharing channels to limit the user's careless behavior. To engage further reproducibility or improvement in this topic, finally, we share the completely code solution which is publicized on the our Github[2].

The remainder of the paper is organized as follows. The next two sections present the background and related work. In Sects. 4 and 5, we introduce the IoT Platform to build a prototype system and its implementation, respectively. In the evaluation section, we discuss our results. Finally, we summarize the main idea of this work and discuss potential directions for future work in the conclusion.

2 Background

2.1 MQTT Protocol

MQTT (Message Queue Telemetry Transport) is a messaging protocol in a publish/subscribe model, using low bandwidth and high reliability. MQTT architecture consists of two main components: Broker and Clients. In which, MQTT Broker is the central server, it is the intersection point of all the connections coming from the client. The main task of the broker is to receive messages from all clients, and then forward them to a specific address. Clients are divided into two groups: publisher and subscriber. The former is the user that publishes messages on a specific topic. Whereas, the latter is the user that subscribe to one or more topics to receive messages going to these topics. In the MQTT protocol, there are 3 levels of QoS as follows:

[2] https://github.com/thanhlam2110/mqtt-sso-kafka.

- QoS-0 (at-most-once): each packet is transmitted to the destination up to once.
- Qos-1 (at-least-once): each packet is passed to the destination at least once, meaning packet iteration can occur.
- QoS-2 (exactly-once): each packet is sent to its destination only once.

2.2 Oauth Protocol and Single Sign-On

Oauth is an authentication mechanism that helps a third party applications to be authorized by the user. The main purpose of the Oauth protocol is to access user resources located on another application. Oauth version 2 is an upgrade version of Oauth version 1, an authentication protocol that allows applications to share a portion of resources with each other without authentication via username and password the traditional way. Thereby, it helps to limit the hassle of having to enter username, password in many places for many applications that are difficult for users to manage.

In Oauth, there are four basic concepts[3]:

- Resource owners: are the users who have the ability to grant access, the owner of the resource that the application wants to get.
- Resource servers: are the places which store resources, capable of handling access requests to protected resources
- Clients: are third-party applications that want to access the shared resource of the owner (i.e., prior to access, the application needs to receive the user's Authorization).
- Authorization servers: are the authentications that check the information the user sent from there, grants access to the application by generating access tokens. Sometimes the same Authorization server is the resource server.

Token is a random code generated by the Authorization server when a request comes from the client. There are two types of tokens, namely the access token and the refresh token. The former is a piece of code used to authenticate access, allowing third-party applications to access user data. This token is sent by the client as a parameter in the request when it is necessary to access the resource in the Resource server. The access token has a valid time (e.g., 30 min, 1 h), when it expired, the client had to send a request to the Authorization server to get the new access token. Whereas, the latter is also generated by Authorization server at the same time with accessed token but with different function. Refresh token is used to get the new access token when it expires, so the validity period is longer than the access token.

Single Sign-On (SSO) is a mechanism that allows users to access multiple applications with just one authentication. SSO simplifies administration by managing user information on a single system instead of multiple separate authentication systems. It makes it easier to manage users when they join or leave an organization [18]. SSO supports many authentication methods such as Oauth, OpenID, SAML, and so on.

[3] https://oauth.net/2/.

2.3 Kafka

Kafka[4] is a distributed messaging system. Kafka is capable of transmitting a large amount of messages in real-time, in case the receiver has not received the message, the message is still stored on the message queue and on the disk to ensure safety. The Kafka architecture includes the main components: producer, consumer, topic, and partition. Kafka producer is a client to publish messages to topics. Data is sent to the partition of the topic stored on the broker. Kafka consumers are clients that subscribe and receive messages from topic, consumers are identified by group names. Many consumers can subscribe to the same topic. Data is transmitted in Kafka by topic, when it is necessary to transmit data for different applications, it is possible to create many different topics. Partition is where to store data of a topic. Each topic can have one or more partitions. On each partition, the data is stored permanently and assigned an ID called offset. In addition, a set of Kafka server is also called a broker and the zookeeper is a service to manage the brokers.

3 Related Work

3.1 Oauth and MQTT

Paul Fremantle et al. [15] used Oauth to enable access control in the MQTT protocol. The paper results show that IoT clients can fully use OAuth token to authenticate with an MQTT broker. The paper demonstrates how to deploy the Web Authorization Tool to create the access token and then embed it in the MQTT client. However, the paper does not cover the control of communication channels, so when the properly authenticated MQTT client is able to subscribe to any topic on the MQTT broker, this creates the risk of data disclosure. The paper presents the combined implementation of Oauth and MQTT for internal communication between MQTT broker and MQTT client in the same organization, but not the possibility of applying for inter-organization communication. Therefore, in our paper, we implement a strict management mechanism for users, devices, and communication channels.

Benjamin Aziz et al. [19] invested Oauth to manage the registration of users and IoT devices. These papers also introduce the concept of Personal Cloud Middleware (PCM) to perform internal communication between the device and a third-party application on behalf of the user. PCM is an MQTT broker that isolates and operates on a Docker or operating system. Each user has their PCM, and this can help limit data loss. However, Benjamin Aziz et al. also said that they do not have a mechanism for revoking PCM when users are no longer using IoT services.

Lam et al. [20] tested the ability to combine MQTT and Oauth through Single Sign On to enhance security. The author proposes a users and things

[4] https://Kafka.apache.org/.

management model - tree model. This architecture has a single user representative for the organization, thus allowing to quickly isolate all users and devices of an organization when the organization is attacked.

3.2 Kafka and MQTT

A.S. Rozik et al. [21] found that the MQTT broker does not provide any buffering mechanism and cannot be extended. When large amounts of data come from a variety of sources, both of these features are essential. In the Sense Egypt IoT platform, A.S. Rozik et al. have used Kafka as an intermediary system to transport messages between the MQTT broker and the rest of the IoT system, which improves the overall performance of the system as well as provides easy scalability.

Moreover, [22] presented Kafka Message Queue and MQTT broker's combined possibilities in Intelligent Transportation System. The deployment model demonstrates the ability to apply to bridge MQTT with Kafka for low latency and handle messages generated by millions of vehicles. They used MQTT Source Connector to move messages from MQTT topic to Kafka Topic and MQTT Sink connector to move messages from Kafka topic to MQTT topic.

In the implementation of the IoT Platform, we also adopt and extend this technique by building APIs that allow users to map their topics.

3.3 Reducing MQTT Energy Consumption

A new approach to the MQTT-SN protocol [23], publishes sensor data (a.k.a the smart gateway selection method) that estimates end-to-end content delay and message loss, during the transmission of content in all levels of QoS. Al-Ali et al. [24] argued that MQTT QoS-2 excluded from test cases due to its large overhead the smart energy management for smart homes and cities. Similarly, Toldinas et al. [10] estimated the energy consumption in transferring data using lightweight MQTT protocol over it different QoS levels (i.e., QoS = 0, QoS = 1, and QoS = 2). However, these works ignore the security and privacy aspects, that is highlighted in [25]. The author investigated the side effects of reliability on IoT communication protocols but they considered MQTT as reliable protocol.

Lam et al. [26] demonstrated an architecture that combines MQTT broker, Single Sign On, and kafka message queue. This combination allows no need to trade-off speed and reliability when communicating with power consumption (this is related to QoS-0 and QoS-2 levels) while still ensuring security. of the system.

4 IoT Platform Proposal

The IoT Platform is a set of APIs combined with system architecture such as Single Sign-On system, Kafka Message Queue, and MQTT broker that provide as following:

- Authenticate information believes the user, thereby granting Oauth access token and refresh token for the user.
- Create and manage user information in the tree model and the unlimited number of user levels created.
- Allowing users to participate in an IoT system capable of creating logical information management of physical devices/ applications and communication channels.
- Allows sending and receiving messages locally when correctly defining a user, a thing, and a specific communication channel.
- Allow users to send and receive messages between two various organizations through the Kafka message queue.

4.1 System Architecture

Figure 1 presents an architectural proposal model of the IoT framework.

- MQTT Broker Cluster is a set of MQTT brokers connected together to distribute the system. MQTT Broker Cluster plays the role of collecting data from IoT devices and transporting control commands from the user to the device according to the MQTT protocol.
- Server Single Sign-On authenticates the user, creates access and refreshes the token according to the Oauth protocol.
- SSO database: contains management information of users, channels and things.
- The Kafka Message Queue stores messages in partitions. This prevents the message from being lost when the receiver has problems.
- MQTT Proxy API provides an interface for users - things to communicate with the MQTT broker.
- API SSO Proxy: provides an interface for users and authenticated parties to communicate with SSO server.

4.2 Software Architecture

To meet the goals set out by the IoT Framework, we provide a number of definitions of the components involved in the system, the design of the database, and the interactions among these components.

4.2.1 Users

Who use IoT services. By constructing the user hierarchy in the model tree with the child's user_parent_id value equal to the parent user's username. Our model allows the creation and management of multiple levels of users and undepending on the organisation's characteristics. This tree-modelled user hierarchy makes suitable for companies, especially when it comes to decentralizing a specific user or changing the state of operation for a series of child users at the same time w.r.t crashes (only the ACTIVE user can request the access token). User information

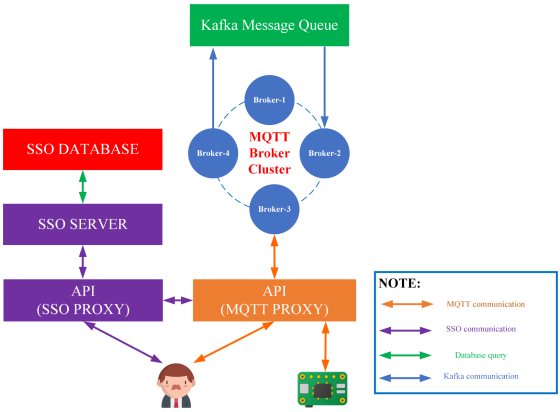

Fig. 1. IoT platform proposal

is generated when registering/using an IoT service or is created by the parent user and provided to their child users.

Each user has a unique user_id value, which conforms to the UUID standard[5] and is managed by the IoT Platform (user is not aware of this value). When publishing or subscribing, the user must pass the access token obtained from the Single Sign-On server, which contains the user_id information and is used as the `clientID` value in the MQTT protocol. In this way, we have enhanced the authentication and authorization mechanisms for the user as well as minimize the risk of a denial-of-service attack when hackers subscribe to a topic with the clientID of the user affecting the accessibility use of the system-wide.

4.2.2 Things

The information about physical devices or applications owned by the user. To create things, the user needs to call the API provided by the IoT Platform, passing in his valid Oauth token. Things information includes two values thing_id and thing_key, conform to the UUID standard and use the equivalent of a username and password values. Only things possess a valid pair of thing_id and thing_key provided by the IoT Platform, that can communicate with the MQTT broker. Since things are not allowed to publish or subscribe directly to the MQTT broker. This layer API validates thing_id and thing_key submitted by things. Similar to the user, this way to create device management information, helps increase security and reduce the risk of denial-of-service attacks.

4.2.3 Channels and Map Things to Channels

In our proposed IoT Platform, channels are the logical concept of managing topics (Kafka and MQTT) that publishes and subscribe to messages by users and things. Users who want to create a channel must call the IoT Platform's API

[5] https://tools.ietf.org/html/rfc4122.

and pass in their valid token Oauth. The channels information has a channel_id value that is unique and follows the standard UUID.

To communicate through the IoT Platform Proposal, the user has to assign things to the channel by calling the API and passing in the existing access token, thing_id, thing_key and channel_id information. The purpose of this process is to allow only one thing with a valid thing_id and thing_key to publish and subscribe to messages on a predefined channel (mapped). From there, avoid the client can subscribe to any topic. This strongly increases the authorization mechanism, which is a flexible way that the original MQTT protocol did not support. The mechanism of assigning things to a channel also enhances security since only things mapped to the channel is able to publish and subscribe to messages on this channel. By implementing this process, users are also able to master their own communication channels.

4.2.4 Publish and Subscribe Message

The process of publishing and subscribing messages via IoT Platform Proposal is described as follows: To perform communication in the IoT Platform, users create two channels with the following roles:

- The channel used to send messages is called "send-channel". The "send-channel" is an MQTT topic and the device will publish the message to this channel through the MQTT Proxy API. Actually, the API uses channel_id but for brevity, we show it by the name of the channel.
- The channel used to receive messages is called a "receive-channel". The "receive-channel" acts as both the Kafka topic and the MQTT Topic.

After creating two channels, we call the API that creates the MQTT source connector, to map the MQTT Topic and the Kafka topic. In this case, we map the "send-channel" and the "receive" channel. This process allows messages from Thing to be sent to "send-channel" to be automatically forwarded to "receive-channel". Right now, the message is being stored in the Kafka message queue, namely, the topic "receive-channel" partition. In order for users to receive messages on this "receive-channel" topic over the MQTT protocol, we implement an MQTT sink connector. This process allows, messages to Kafka topic "receive-channel" are received by users who subscribe to the MQTT topic "receive-channel".

During Publish and Subscribe message, the MQTT Proxy API also calls the authentication service from the SSO Proxy API to validate the token, check for thing_id, thing_key, and assign things to the channel. The process of publishing and subscribing to the message is shown in Fig. 2.

5 Implement

5.1 Database

As explained in Sect. 4.2, the IoT Platform allows to manage users, things, channels information and implement map things to channel. In practical implementation, we use MongoDB as a NoSQL database management system, and to

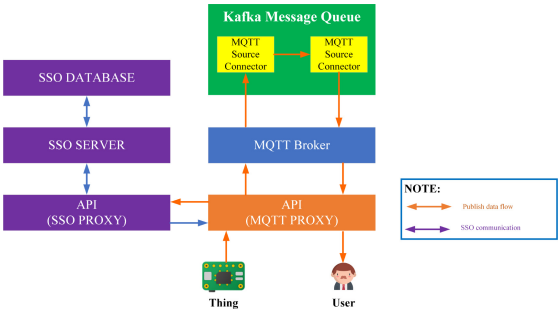

Fig. 2. Process publish a message to the public

implement the model tree (select, update, delete) outlined in Sect. 4.2.1, we used Aggregation techniques are provided by MongoDB[6].

5.2 Single Sign-On

In the prototype system, we use the open-source CAS Apereo[7] to provide the Single Sign-On service. CAS Apereo supports many protocols for implementing single sign-on services such as Oauth, SAML, OpenID, and so on. The IoT Platform protocol used to communicate with the Single Sign-On server is Oauth. In our implementation, the clients do not interact directly with the CAS server but instead, we provide the set of APIs through the SSO Proxy layer (API SSO Proxy layer).

5.3 Mosquitto MQTT Broker

Mosquitto is an open-source to implement an MQTT broker that allows to transmit and receive data according to MQTT protocol. Mosquitto is also part of the Eclipse Foundation[8]. Mosquitto is very light and has the advantages of fast data transfer and processing speed, high stability.

5.4 Prototype Model

The Prototype system we deployed on Amazon EC2[9] infrastructure consists of three servers as shown in Table 1.

According to the MQTT protocol, the first server deploys the Mosquitto MQTT Broker service in the Prototype model to collect and forward messages. The second server, deploying MQTT Broker 1b service for public communication.

[6] https://docs.mongodb.com/manual/reference/operator/aggregation/graphLookup/.
[7] https://apereo.github.io/cas/6.3.x/index.html.
[8] The project iot.eclipse.org (https://iot.eclipse.org/).
[9] https://aws.amazon.com/ec2/pricing/.

Table 1. The list of servers in the prototype

Server	Role	Server configuration
MQTT Broker	Deploy service MQTT broker Deploy API MQTT Proxy	CPU 1 RAM 1GB
Kafka server	Deploy the Kafka service	CPU 1 RAM 1GB
SSO server	Deploy Single Sign-On service Deploy the SSO database service Deploy the service API SSO Proxy	CPU 2 RAM 1GB

Also, on the first server, we deploy the MQTT Proxy API service to provide interfaces for users and things that communicate with MQTT brokers through the API. Whereas, the second server implements the Single Sign-On service, which creates access to tokens and refreshes tokens for users according to the Oauth protocol. Besides, the SSO Proxy API service provides interfaces for users, things, and the MQTT Proxy API performs SSO server communication through the API. In addition, the second server also has MongoDB installed to store user information, thing, and channel, presented in Sect. 4.2.

The Kafka service, deployed on a third server, acts as a message queue to store messages exchanged through the MQTT broker. This ensures that when things or users experience interruptions, notifications are not lost as well as limit infinite timeouts or packet repetitions.

6 Evaluation

Our proposed model has two goals. The first is to strengthen the security mechanism for MQTT Protocol. The second is to build a mechanism to take advantage of the advantages of MQTT QoS-0 rather than QoS-2, such as transmission speed, bandwidth and low energy consumption while ensuring reliability when transmitting packets. To test these goals, we execute the following scenarios.

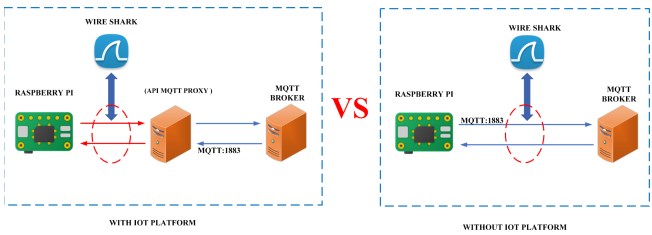

Fig. 3. The first scenario test model

The first scenario uses Wireshark software[10] to capture packets when things send messages to the MQTT broker in two cases. In the first case, things do connect and publish the message directly to the MQTT broker. In the second case, things do connect and publish the message through the MQTT Proxy API class provided by the IoT Platform Proposal. The first scenario model is shown in Fig. 3. The outcomes captured of the first scenario from the Wireshark software are shown in Fig. 4

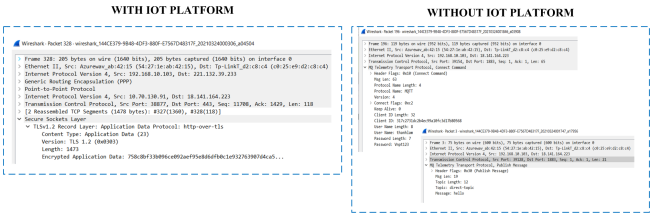

Fig. 4. Capture message of the two cases (with and without IoT Platform)

The resulting snapshot shows that publish the message directly to the MQTT broker, it takes two steps: "Connect" and "Publish" in the case of things. Performing packet capture can obtain all `ClientID`, username, and password in the "Connect" packet and MQTT topic along with the message content in the "Publish" packet. When things publish a message via IoT Platform Proposal, it only takes one step of "Publish". The content of the message is encrypted because the IoT Platform supports TLS. After the IoT Platform receives the message, it will use the information in the message such as access token, channel_id, things_id and things_key to authenticate the request, make a connection to the MQTT broker, and transport the packet according to the received topic. From here, the use of the IoT Platform Proposal significantly improves the security risks of the MQTT protocol.

Whereas, the second scenario examines the packet transfer rate, throughput, and the number of things that can publish messages simultaneously in three test-cases:

- **Test 1**: things connect directly to the MQTT broker and do publishing messages according to QoS-2.
- **Test 2**: things connect to the MQTT broker through the IoT Platform Proposal and do publish the message according to QoS-0. In this case, do not apply security checks such as: validity of access token, check assigning things to channel.
- **Test 3**: things connect to MQTT broker via IoT Platform Proposal, publish messages according to QoS-0 and apply security check function.

[10] Wireshark (https://www.wireshark.org/) is a network packet analyzer software (a network packet analyzer) capable of monitoring and monitoring packets in real-time.

The test we use is Apache Jmeter[11] that is fully written in Java language, and usable for performance testing on static resources, dynamic resources and Web applications. This is used to simulate a large number of virtual users, large requests on a server, a group of server/network, an object to test tolerance, load test or analyze the response time. Jmeter firstly makes the requests and sends them to the server according to the predefined method. Then, it receives responses from the server, collects them and displays information in the report (report). Jmeter has many report parameters, but we consider in two parameters: throughput and error, with throughput (request/s), which is the number of requests processed by the server per second and error (%) is the percentage of requests that fail to total requests. Test results are shown in Table 2.

Table 2. Test results of the second scenario

	Result	100 CCU	200 CCU	300 CCU	400 CCU	500 CCU
Test 1	Throughput	48.8	97.4	139.1	193.4	–
	Error	0	0	0	47%	–
Test 2	Throughput	95.3	189.6	312.5	226.6	208.7
	Error	0	0	0	0	0
Test 3	Throughput	48.8	50.1	51.4	50	38.8
	Error	0	0	0	0	0

Table 2 shown that the use of QoS-2 helps ensure the packet is transmitted to the destination and not duplicated in Test 1. However, to achieve this, we have to trade system performance such as lower throughput and higher error rate since the server has to handle more when sending messages in QoS-2. We noted the effectiveness of MQTT and Kafka Message Queue's combination w.r.t in Test 2. Using QoS-0 for faster processing speed and throughput was much higher than using QoS-2, the packets, in this case, have still send the guaranteed and did not duplication thanks to Kafka's capabilities. The third test is completely acceptable as our IoT Platform proposal has added the test and validation mechanisms as outlined in Sect. 4.2.

To develop a larger scenario and increase the number of devices/users authorized quickly, other security issues such as security, privacy, availability for objects are still the challenges. For the security aspect, further works will be deployed in different scenarios like healthcare environment [27–29], cash on delivery [30,31]. For the privacy aspect, we will exploit attribute-based access control (ABAC) [32,33] to manage the authorization process of the IoT Platform via the dynamic policy approach [34–36]. Besides, we will apply the blockchain benefit to improve the availability issues [37–39]. Finally, we eliminate reliance on MQTT Broker for the data collection process and routing and building a more proactive method of data collection for the low-energy devices.

[11] Apache Jmeter https://jmeter.apache.org/.

7 Conclusion

In this paper, we propose a combination method of Oauth protocol, Single Sign-On, user management model, thing and channel to improve the security of MQTT protocol. The method of assigning things to the channel helps the system strictly manage the communication channels to minimize the careless behavior of users when sharing data. Besides, MQTT and Kafka Message Queue's combination allows our approach to accepting low-resource devices such as low communication speeds, low energy consumption, and low transmission bandwidths while still providing reliability in transmitting. The evaluation section also shows the effectiveness of the proposal IoT Platform.

References

1. Alam, T.: A reliable communication framework and its use in internet of things (IoT). CSEIT1835111/Received **10**, 450–456 (2018)
2. Chou, T.: Precision-Principles, Practices and Solutions for the Internet of Things. McGraw-Hill Education, New York (2017)
3. Karagiannis, V., et al.: A survey on application layer protocols for the internet of things. Trans. IoT Cloud Comput. **3**(1), 11–17 (2015)
4. Niruntasukrat, A., et al.: Authorization mechanism for MQTT-based internet of things. In: 2016 IEEE International Conference on Communications Workshops (ICC), pp. 290–295. IEEE (2016)
5. Jaikar, S.P., Iyer, K.R.: A survey of messaging protocols for IOT systems. Int. J. Adv. Manag. Technol. Eng. Sci. **8**(II), 510–514 (2018)
6. Çorak, B.H., et al.: Comparative analysis of IoT communication protocols. In: 2018 International Symposium on Networks, Computers and Communications (ISNCC), pp. 1–6. IEEE (2018)
7. Hillar, G.C.: MQTT Essentials-A Lightweight IoT Protocol. Packt Publishing Ltd, Birmingham (2017)
8. Martı, M., Garcia-Rubio, C., Campo, C.: Performance evaluation of CoAP and MQTT_SN in an IoT environment. In: Multidisciplinary Digital Publishing Institute Proceedings, vol. 31, no. 1, p. 49 (2019)
9. Lee, S., et al.: Correlation analysis of MQTT loss and delay according to QoS level. In: The International Conference on Information Networking 2013 (ICOIN), pp. 714–717. IEEE (2013)
10. Toldinas, J., et al.: MQTT quality of service versus energy consumption. In: 2019 23rd International Conference Electronics, pp. 1–4. IEEE (2019)
11. Mendez Mena, D., Papapanagiotou, I., Yang, B.: Internet of things: survey on security. Inf. Secur. J. Glob. Perspect. **27**(3), 162–182 (2018)
12. Lundgren, L.: Light-Weight Protocol! Serious Equipment! Critical Implications! In: Defcon 24 (2016)
13. Anthraper, J.J., Kotak, J.: Security, privacy and forensic concern of MQTT protocol. In: Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM), Amity University Rajasthan, Jaipur-India (2019)

14. Zaidi, N., Kaushik, H., Bablani, D., Bansal, R., Kumar, P.: A study of exposure of IoT devices in India: using Shodan search engine. In: Bhateja, V., Nguyen, B.L., Nguyen, N.G., Satapathy, S.C., Le, D.-N. (eds.) Information Systems Design and Intelligent Applications. AISC, vol. 672, pp. 1044–1053. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-7512-4_105
15. Fremantle, P., et al.: Federated identity and access management for the internet of things. In: 2014 International Workshop on Secure Internet of Things, pp. 10–17. IEEE (2014)
16. Tawalbeh, L., et al.: IoT Privacy and security: challenges and solutions. Appl. Sci. **10**(12), 4102 (2020)
17. Subahi, A., Theodorakopoulos, G.: Detecting IoT user behavior and sensitive information in encrypted IoT-app traffic. Sensors **19**(21), 4777 (2019)
18. Radha, V., Reddy, D.H.: A survey on single sign-on techniques. Procedia Technol. **4**, 134–139 (2012)
19. Fremantle, P., Aziz, B.: Oauthing: privacy-enhancing federation for the internet of things. In: 2016 Cloudification of the Internet of Things (CIoT), pp. 1–6. IEEE (2016)
20. Nguyen, L., Thanh, T., et al.: Toward a unique IoT network via single sign-on protocol and message queue. In: International Conference on Computer Information Systems and Industrial Management (2021)
21. Rozik, A.S., Tolba, A.S., El-Dosuky, M.A.: Design and implementation of the sense Egypt platform for real-time analysis of IoT data streams. Adv. Internet Things **6**(4), 65–91 (2016)
22. Hugo, Å., et al.: Bridging MQTT and Kafka to support C-ITS: a feasibility study. In: 2020 21st IEEE International Conference on Mobile Data Management (MDM), pp. 371–376. IEEE (2020)
23. Roy, D.G., et al.: Application-aware end-to-end delay and message loss estimation in Internet of Things (IoT)-MQTT-SN protocols. Future Gener. Comput. Syst. **89**, 300–316 (2018)
24. Al-Ali, A.-R., et al.: A smart home energy management system using IoT and big data analytics approach. IEEE Trans. Consum. Electron. **63**(4), 426–434 (2017)
25. Safaei, B., et al.: Reliability side-effects in Internet of Things application layer protocols. In: 2017 2nd International Conference on System Reliability and Safety (ICSRS), pp. 207–212. IEEE (2017)
26. Thanh, L.N.T., et al.: UIP2SOP: a unique IoT network applying single sign-on and message queue protocol. IJACSA **12**(6) (2021)
27. Son, H.X., Chen, E.: Towards a fine-grained access control mechanism for privacy protection and policy conflict resolution. Int. J. Adv. Comput. Sci. Appl. **10**(2), 507–516 (2019)
28. Duong-Trung, N., et al.: Smart care: integrating blockchain technology into the design of patient-centered healthcare systems. In: International Conference on Cryptography, Security and Privacy, pp. 105–109 (2020)
29. Duong-Trung, N., et al.: On components of a patient-centered healthcare system using smart contract. In: Proceedings of the International Conference on Cryptography, Security and Privacy, pp. 31–35 (2020)
30. Le, H.T., et al.: Introducing multi shippers mechanism for decentralized cash on delivery system. Money **10**(6) (2019)
31. Le, N.T.T., et al.: Assuring non-fraudulent transactions in cash on delivery by introducing double smart contracts. IJACSA **10**(5), 677–684 (2019)

32. Hoang, N.M., Son, H.X.: A dynamic solution for finegrained policy conflict resolution. In: The International Conference on Cryptography, Security and Privacy, pp. 116–120 (2019)
33. Son, H.X., Hoang, N.M.: A novel attribute-based access control system for fine-grained privacy protection. In: The International Conference on Cryptography, Security and Privacy, pp. 76–80 (2019)
34. Xuan, S.H., et al.: Rew-XAC: an approach to rewriting request for elastic ABAC enforcement with dynamic policies In: International Conference on Advanced Computing and Applications, pp. 25–31. IEEE (2016)
35. Thi, Q.N.T., Dang, T.K., Van, H.L., Son, H.X.: Using JSON to specify privacy preserving-enabled attribute-based access control policies. In: Wang, G., Atiquzzaman, M., Yan, Z., Choo, K.-K.R. (eds.) SpaCCS 2017. LNCS, vol. 10656, pp. 561–570. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72389-1_44
36. Son, H.X., Dang, T.K., Massacci, F.: REW-SMT: a new approach for rewriting XACML request with dynamic big data security policies. In: Wang, G., Atiquzzaman, M., Yan, Z., Choo, K.-K.R. (eds.) SpaCCS 2017. LNCS, vol. 10656, pp. 501–515. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72389-1_40
37. Ha, X.S., et al.: DeM-COD: novel access-control-based cash on delivery mechanism for decentralized marketplace. In: International Conference on Trust, Security and Privacy in Computing and Communications, pp. 71–78. IEEE (2020)
38. Ha, X.S., Le, T.H., Phan, T.T., Nguyen, H.H.D., Vo, H.K., Duong-Trung, N.: Scrutinizing trust and transparency in cash on delivery systems. In: Wang, G., Chen, B., Li, W., Di Pietro, R., Yan, X., Han, H. (eds.) SpaCCS 2020. LNCS, vol. 12382, pp. 214–227. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-68851-6_15
39. Son, H.X., Le, T.H., Quynh, N.T.T., Huy, H.N.D., Duong-Trung, N., Luong, H.H.: Toward a blockchain-based technology in dealing with emergencies in patient-centered healthcare systems. In: Bouzefrane, S., Laurent, M., Boumerdassi, S., Renault, E. (eds.) MSPN 2020. LNCS, vol. 12605, pp. 44–56. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67550-9_4