Secure Push notification service based on MQTT Protocol for mobile platforms

Carlos Silva[1, 2], Renato Toasa[1, 4], H. David Martinez[3], Jorge Veloz[2] and Cristian Gallardo[4]

[1]School of Technology and Management, Polytechnic Institute of Leiria, Portugal,
{2162317, 2162342}@my.ipleiria.pt
[2]Universidad Técnica de Manabí, Ecuador,
jveloz@utm.edu.ec
[3]Universidad de Guayaquil, Ecuador,
hector.martinezvi@ug.edu.ec
[4]Universidad Técnica de Ambato, Ecuador,
cmgallardop@gmail.com

Abstract. Push notification is conceptually an event-based mechanism where remote servers push events to smartphone client apps when these occur.
The communication between the MQTT client and the remote server has been performed via TCP by default, causing certain level of uncertainty in the obtaining data, which is necessary to enhance an adequate configuration of the server with the aim of generating security in the transmission and integrity of the data.
This paper present the secure MQTT services implement on android and iOS clients, developing a push notification service that can be used to solve the problem of instant pushing several messages on a company. It also details the different areas in which the MQTT protocol could be applied.

Keywords: Push notification, service, secure, MQTT, android, iOS, SSL.

1. Introduction

Smartphone adoption continues to permeate society, majority of people of the world now use a smartphone. The people are increasingly dependent on their mobile phones to get real-time messages, so instant messaging is particularly important, this comes in conjunction with push / notification systems [1].

Currently, the push/notification technology has become the most popular network of technology, many companies are aiming at the technology for research, and launched their own products, and apply this technology to many operating systems [2].

It is an irrefutable fact that in the last few years mobile phones have revolutionized the whole world, and part of this important world trend has involved changing the way people communicate with one another; one of the elements of this change has been instant messaging service, for developer push notification services is possible to use various protocol such as XMPP, Websockets, CoAP and MQTT. These protocols are used in each different situation. In particular, MQTT protocol was designed to work on low-power devices nicely as a light-weight protocol and has been used in many IoT devices and instant messaging systems [3].

MQTT protocol become our first choice because of its obvious advantages. It is an open, simple, lightweight messaging protocol and easy to implement. Initially it was designed to connect large numbers of remote sensors and control devices. The protocol has been applied in a variety of embedded systems. For example: hospitals use this protocol to communicate with pacemakers and other medical equipment supplier. Oil and gas companies use it to monitor oil pipeline thousands of miles away [4].

A mobile application that uses MQTT sends and receives messages by calling an MQTT library. Messages are exchanged through an MQTT messaging server. The MQTT client and server handle the complexities of delivering messages reliably to the mobile application and keep the price of network management under control [5]. Also, the goal of the protocol is to provide a reliable service that consumes few resources. So that's why in the protocol itself are only a few security mechanisms clearly specified. But in all implementations with Mqtt other security standards are used, like SSL/TLS for transport security, for this reason It can be stated that the data sent through Mqtt are protected[6].

MQTT applications run on mobile devices, such as smartphones and tablets. MQTT is also used in telemetry to receive data from sensors and to control them remotely. For mobile devices and sensors, MQTT offers a highly scalable publication / subscription protocol with secure delivery

In this paper, the secure service based on the MQTT protocol in android and iOS platforms is proposed to achieve instant messaging push. The process of subscription of the topic and the publication of messages in that topic is done in real time, in this way the message push / notification can be optimized and innovated.

The rest of the paper is organized as follows. Section II describes the context and state of the art. Section III presents the MQTT Protocol functionality, while the secure communication is shown in Section IV. Next, in Section V the Implementation of Mqtt Protocol is detailed, Next, in Section VI the applicable areas of Mqtt, in section VII the test and results and in section VIII the conclusions are presented.

2. Context and State of the Art

Currently there are few reports of MQTT usage to develop system based in mobile platform, the search for articles regarding this subject has revealed itself unsuccessful, but a wider search helped to find cases of MQTT, that are detailed below:

In the work [7] they created a system for acquiring and transmitting data via Bluetooth to an android mobile platform using the MQTT library for the communication with a remote server, where the data is stored in a database and visualization of Pulse Oximeter. In this work the client authentication via SSL was not implemented, the communication between the client and the remote server has been realized via TCP, so the security in the transmission and the medical data integrity is not warranted, causing a certain level of uncertainty in the obtaining data, because the information can be susceptible to capture and modification during the transmission process.

There are also some industry works on mobile push notification services. Currently the push notification service providers can be divided into two categories. Some of the services/products provide basic push notification service, and the other services/products provide enhanced push notification services [8].

The basic push notification service providers, such as Apple Push Notification Service [9], Microsoft Push Notification Services [11], can help users to deliver notifications through mobile devices. , these services only provide basic features for messaging service and cannot provide rich user experience. For example, Apple Push Notification Service only support the iOS platform. It makes the user inconvenient when the user need to push a notification to various platforms in most of the cases. However, the service Firebase Cloud Messaging [10], Amazon Device Messaging [12] and Facebook Push Notification [13] support pushing a notification to users with special context properties, these services support more one specific platform. MQTT can be implemented on platforms like android, iOS, python and other platforms and there is documentation on internet about the implementation and how to correct errors that occur during the addition to an existing project.

MQTT relies on TCP as transport protocol, which means by default the connection does not use an encrypted communication. To encrypt the whole MQTT communication, most many MQTT brokers allow to use TLS instead of plain TCP, when has been using the username and password fields of the MQTT connect packet for authentication and authorization mechanisms, the administrator should strongly consider using TLS in MQTT broker. Is the user's responsibility to address security issues for MQTT and MQTT-SN. In this direction, it is suggested to enable security for MQTT by envisaging SSL/TLS with certificates and session key management. [14].

3. MQTT Protocol functionality

MQTT protocol is the message push protocol released by IBM. It was designed to transfer a message reliably under the low-bandwidth network condition and long network delay. Currently, everything that is a web of things software company, Amazon IoT use MQTT protocol [4].

The MQTT push protocol defines the transmission Quality of Service (QoS) levels, which is an agreement between sender and receiver of a single message regarding the guarantees of delivery of message. There are three levels of QoS in MQTT [15, 16].

- QoS 0: At most once. Send a message at most once and do not guarantee to deliver a message.
- QoS 1: At least once. Send a message at least once and it is possible to deliver a message more than once.
- QoS 2: Exactly once Send a message exactly once with 4 way handshaking.

MQTT protocol consists of subscriber, publish and message broker. A client who wants to get the massage belong to certain topic subscribes topic to a message broker server. If another client publishes the message with the certain topic, then a message broker server republish the message to the subscriber who already subscribes with the certain topic. Topic is a message string to filter messages for each client and it consists of one or more topic levels. Each levels is distinguished by slash (/) and the structure looks like tree structure [17].

The Publishing/Subscribing Message Model. The design is based on publishing/subscribing messaging model. In this model, the publisher and subscriber are both clients, the message destination is called theme. By connecting to the message broker, they transfer data across the network. Publishers send a specific topic of messages to message broker, subscribers subscribe specific news topics to the message broker, and the connection between the subscriber and publishers managed by the message broker. When the message broker receives the published messages, it delivers the message to subscriber. Publishing/subscribing messaging model allows multiple providers to publish messages to the same topic. It also allows multiple users to subscribe messages with a subject [18]. Then the message broker will broadcast to different subscribers, the figure 1 show the publish - subscribe architecture

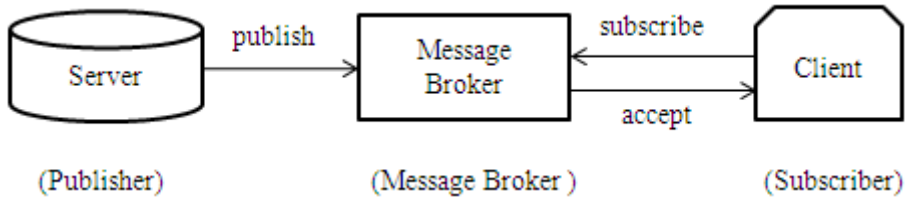

Fig. 1. Overall Architecture Diagram.

As a message broker, it completes message routing function, receiving the message sent by the server, and then forwarded. Subscribers subscribe to the interesting topics, to submit information to the Message Broker, and to maintain a persistent connection. If publisher check into a new message, this message will be released by topic category. Depending on the topic, the Message Agent receives the message as well as the client's subscription, and releases this news to the corresponding phone, the figure 2 show the push process in the Mqtt protocol [17].

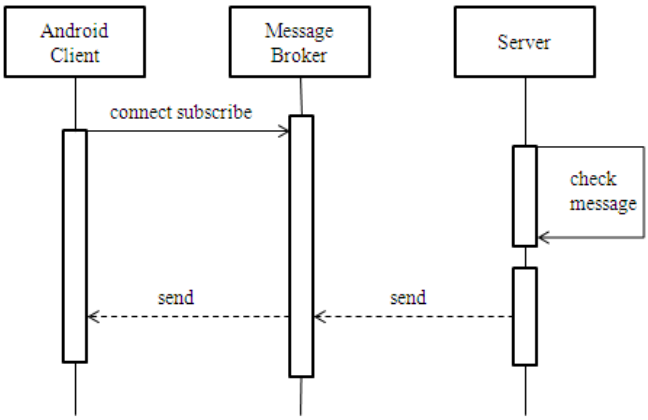

Fig 2. Push Process.

The process in detail is: First, connecting MQTT. The client specifies the message broker host 's IP address and port number assigned by MQTT, establishes a connection and creates MQTT client object. MQTT object subscribes to a topic, requiring both subscription array parameter and service quality array parameter.

Second, releasing Information to the Agency. Interface server directly using the API functions provided by MQTT, requires theme, message, service quality and other parameters. Send push messages to the Message Broker.

Third, receiving a Message from the Agent. To make sure that the subscriber can receive messages, you must create a callback function and register in MQTT client. Callback function informs system that topic message has arrived at the client.

4. Secure communication

There is a drawback when using MQTT over TLS: security comes at a cost in terms of CPU usage and communication overhead. While the additional CPU usage is typically negligible on the broker, it can be a problem for very constrained devices which are not designed for compute intensive tasks. Techniques like Session

Resumption can drastically improve the TLS performance.TLS session resumption techniques allow to reuse an already negotiated TLS session after reconnecting to the server, so the client and server don't need to do the full TLS handshake again. It's important to note, that not all TLS libraries implement all session resumption techniques, some don't even implement any session resumption mechanism. [19]

The Port 8883 is standardized for a secured MQTT connection. The standardized name at IANA is "secure-mqtt" and port 8883 is exclusively reserved for MQTT over TLS.

Security in MQTT is divided in multiple layers. Each layer prevents different kind of attacks. The goal of the protocol is to provide a really lightweight and easy to use communication protocol for the internet of things. So that's why in the protocol itself are only a few security mechanisms clearly specified. But in all common implementations other security standards are used, like SSL/TLS for transport security [19].

4.1. Network Level

Using a physically secure network or VPN as foundation for any communication between clients and broker is one way to provide a secure and trustworthy connection. This would be suitable for gateway applications, where the gateway is connected to devices on the one hand and with the broker over VPN on the other side.

4.2. Transport Level

When the goal is to provide confidentiality in most cases TLS/SSL is being used for transport encryption. It provides a secure and proven way to make sure nobody can read along and even authenticate both sides, when using client certification authentication. We will also cover in detail the feasibility of TLS on constrained devices.

4.3. Application Level

On the transport level it can be ensured that the communication is encrypted and the identity is authenticated. The MQTT protocol provides a client identifier and username/password credentials, which can also be used to authenticate devices on the application level. These properties are provided by the protocol itself. When it comes to authorization or what each device is allowed to do, it lays in the hand of the broker implementation, how to handle it. Another possibility is to use payload encryption on

the application level in order to make the transmitted information secure even without having a full-fledged transport encryption. [19]

Connections between the MQTT client and the queue manager are always initiated by the MQTT client. The MQTT client is always the SSL client. Client authentication of the server and server authentication of the MQTT client are both optional.

To implement SSL in MQTT brokers, it is necessary to perform the following actions:

- Generate the server certificates
- Configure SSL in MQTT Broker
- Configure MQTT Client for use SSL/TLS

4.4. MQTT Client configuration

To authenticate the MQTT client using SSL, the client connects to a telemetry channel using SSL. It must specify a TCP port that corresponds to a telemetry channel that is configured to authenticate SSL clients. For example, at the client:

```
MQTTClient mqttClient = new
MqttClient("ssl://www.example.org:8883","clientId1");
mqttClient.connect();
```

Add the digital certificate of the client, signed either using the private key of the client, or by a CA, to the password protected keystore on the client. If the certificate has a keychain, you can add the certificates from the keychain to the store. When the server verifies the client certificate, it uses the certificates sent by the client to match against certificates in its keystore. It is looking for the first match in the key chain with a certificate that it has. The remainder of the keychain is ignored. The MQTT client sends all the certificates in its keystore to the server. If the server authenticates any of the key chains the client sends, then the client is authenticated [4].

5. Implementation of Mqtt Protocol

5.1. General Architecture

The service proposed in this article allows the user to enter the server to be connected and define the port, this is done because some companies have their own servers and ports enabled, and do not require the free servers offered by Mqtt.

Once it is accessed, the user can subscribe the channel or topic in which the messages will be sent, and post messages in the defined channel. All these messages will be available; to be analyzed according to the needs of the people or companies that use the service. Figure 3 shows the service MQTT architecture and the push notification system.

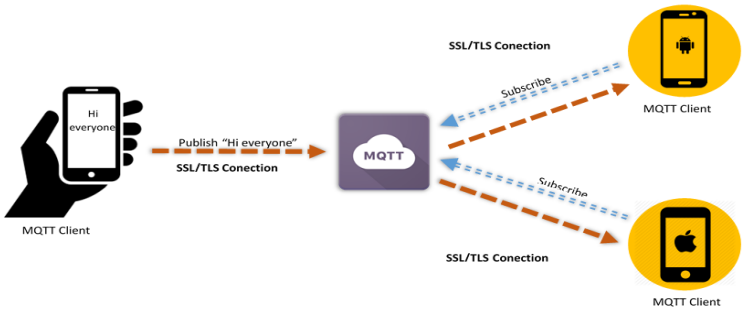

Fig. 3. Service MQTT architecture.

5.2. Server Configuration

The server used is Ubuntu 16.04 of 32 bits, for install Encrypted Certificates we'll install the official client from an Ubuntu PPA, or Personal Package Archive with:

```
sudo add-apt-repository ppa:certbot/certbot
apt-get install certbot
```

Certbot needs to answer a cryptographic challenge issued by the Let's Encrypt API in order to prove we control our domain. It uses ports 80 (HTTP) and/or 443 (HTTPS) to accomplish this. We'll only use port 80, so let's allow incoming traffic on that port now:

```
sudo ufw allow http
```

Now run Certbot to get the certificate. Use the --standalone option to tell Certbot to handle the HTTP challenge request on its own, and --standalone-supported-challenges http-01 limits the communication to port 80. -d is used to specify the domain you'd

like a certificate for, and certonly tells Certbot to just retrieve the certificate without doing any other configuration steps.

```
sudo certbot certonly --standalone --standalone-supported-challenges http-01 -d localhost
```

When running the command, you will be prompted to enter an email address and agree to the terms of service. After doing so, you should see a message telling you the process was successful and where your certificates are stored.

To enable SSL encryption, we need to tell Mosquitto where our Let's Encrypt certificates are stored. Open up the configuration file we previously started:

```
sudo nano /etc/mosquitto/conf.d/default.conf
listener 1883 localhost
listener 8883
certfile  /etc/letsencrypt/live/localhost/server.crt
cafile    /etc/letsencrypt/live/localhost/server.crt
keyfile /etc/letsencrypt/live/localhost/server.key
```

We're adding two separate listener blocks to the config. The first, listener 1883 localhost, updates the default MQTT listener on port 1883, which is what we've been connecting to so far. 1883 is the standard unencrypted MQTT port. Listener 8883 sets up an encrypted listener on port 8883. This is the standard port for MQTT + SSL, often referred to as MQTTS. The next three lines, certfile, cafile, and keyfile, all point Mosquitto to the appropriate Let's Encrypt files to set up the encrypted connections.

Since mqtt works with several users, it is necessary to configure Mqtt to handle connections by user and password, mosquitto includes a utility to generate a special password file called mosquitto_passwd.

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd user
```

Now we'll open up a new configuration file for Mosquitto y add the next lines For use this password file to require logins for all connections

```
sudo nano /etc/mosquitto/conf.d/default.conf
allow_anonymous false
password_file /etc/mosquitto/passwd
```

Now mqtt allows connections by user and password, with encrypted messages. The operation is shown below:

```
mosquitto_sub -h localhost -t test -u "user" -P
"password"

mosquitto_pub -h localhost -t test -m "hello" -p
8883    --cafile    /etc/ssl/certs/server.crt    -u
"ipleiria" -P "ipleiria"

mosquitto_sub -h localhost -t test -u "user1" -P
"secretpassword"
hello
```

6. Applicable Areas

The applications for the MQTT protocol are infinite; here is a list of some applicable areas and their respective example:

6.1. Personal and Home(IoT)

The IoT is already connecting computing devices, appliances, humans and other living beings through the Internet. MQTT clients are very simple to implement.

MQTT can be used in this area as communication technology between devices, for example: it is possible to send information, in this case a power command to all devices in real time when a user arrives at the smart house, plus the flexibility of this protocol allows each device to be controlled individually because the MQTT architecture is based on publish subscribe architecture.

6.2. Wireless sensor networks

Wireless sensor networks (WSN), sometimes called wireless sensor and actuator networks (WSAN) are spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, pressure, etc. MQTT protocol can be used in this field, and has many advantages over other protocols, because it is a

protocol that consumes very little bandwidth and can be used in most embedded devices with few resources (CPU, RAM, etc.).

6.3. Health field

For example, in sensing and monitoring application, in health field is necessary the obtaining and presentation of medical data in real time. For example, the body sensors that have implemented an MQTT Client can emit information of the state of the patient to the devices of all doctors from different areas such as cardiology, allergology, infectology, neurology or only to a specific doctor, all this is possible because the MQTT protocol is light and based on a publisher / subscriber architecture.

6.4. Enterprise

In the companies can be used for the development of internal messaging application (Push notification), for real-time communication with company workers, especially in health care companies such as renal dialysis centers, intensive care centers , among others, in those places where notifying certain doctors about an event is very important issue when the time is a decisive factor.

6.5. Web and social Networks

The MQTT protocol can be used for the development of mobile applications that allow users to interact with each other through text messages, currently there are many instant messaging services such as facebook messenger, whatsApp, telegram and instagram, snapchat, among others, one of the problems they have is that messages and information from users using these services is stored in international companies, generating at some point less privacy for users.

7. Test and Results

In order to perform the tests of the service proposed in this research, two mobile applications prototype were developed, one for android platform and the other for iOS platform, the figure 4 show the interfaces of applications.

Functionality tests were performed on these platforms in order to verify the operation of the service or push / notification with Mqtt. Obtaining the following results:

The tests were performed in virtual machine in VMware Fusion 8.5.7 with the following characteristics:

CPU : 1 CPU virtual

RAM: 1 GB

Operating System: Ubuntu 16.04.2 LTS 32bits

The MQTT server used was mosquitto version 1.4.8.

7.1. Mobile Application

The first mobile application version has been developed for Android and iOS operating system, the iOS application has been developed in Xcode with Swift 3.0.2, Xcode is Apple's official integrated development environment (IDE), and uses the Swift programming language, which is focused on developing native applications for iOS and Mac OS X [20].Has been used the SwiftMQTT library for the application prototype [21]. The Figure 4 shows the main interfaces of the iOS application.

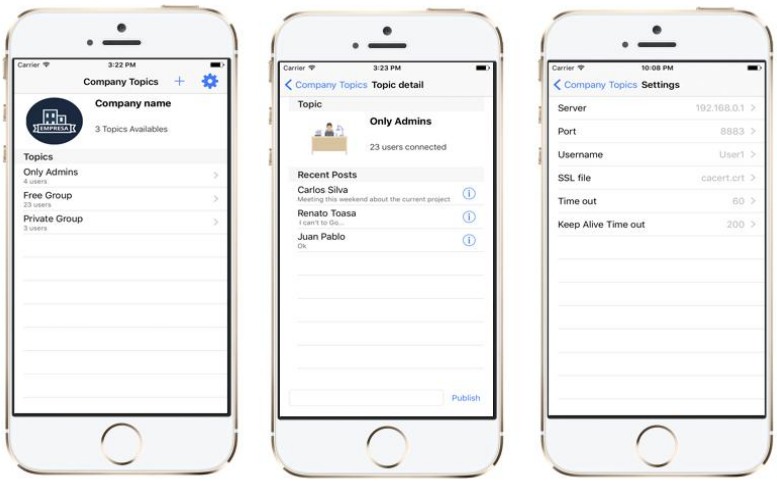

Fig 4. iOS application screens.

The application for Android operating systems is developed in Android Studio 2.3 with JAVA programming language and is compatible with Android 4.1 (Jelly Bean) and above, has been used the Eclipse Paho libraries. The Paho Java Client is an MQTT client library written in Java for developing applications that run on the JVM or other Java compatible platforms such as Android.[22].The Figure 5 shows the main interfaces of Android application.

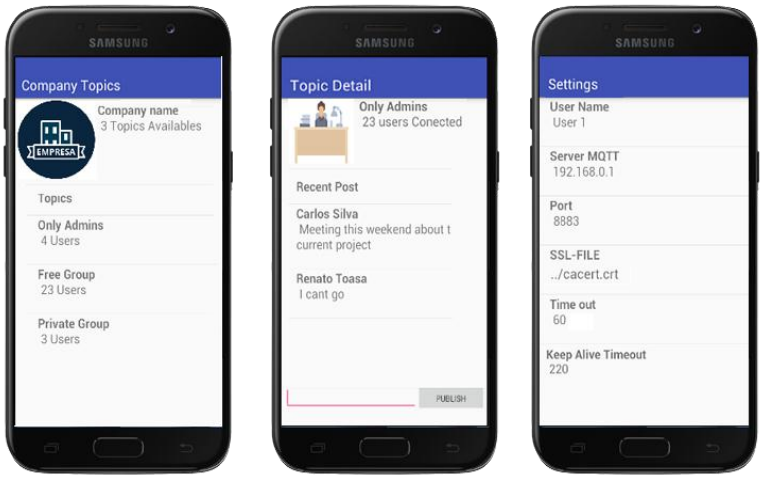

Fig. 5. Android application screens.

The functionality test was performed on a local server with 15 users, subscribed to the same topic, and these made publications, the service worked correctly without delays or lost messages. The Mqtt protocol is designed to support thousands of connected users; this will depend on how the service is implemented.

Performance tests were performed on the mosquito server, sending messages to the server using certificates and not using it, showing that in a total 100.000 messages using certificates for communication between the client and the server decreases server performance between 60% and 65 % in the table 1 the results can be visualized.

During performance tests using certificates for communication between the MQTT server and the client, the server uses six times more CPU and twice the amount of RAM to perform the operations.

Table 1. Performance test on mosquito server

Quantity	Connection Type	Average message time	Total Time	Loss rate
100.000 messages	Normal	11 mls – 12 mls	20 min 3s 911mls	0.00%
100.000 messages	SSL	19 mls – 20 mls	33 min 16s 256mls	0.004%

8. Conclusions

Can be assured that MQTT is the protocol has used to provide new and revolutionary performance. It opens new areas for messaging use cases. As MQTT specializes in low-bandwidth, high-latency environments, it is considered an ideal protocol for machine-to-machine (M2M) communication. The MQTT design makes it appealing for the exponential emerging Internet of Things (IoT) market and new messaging systems.

During the implementation of the service, it is necessary to enhance an adequate configuration of the server with the aim of generating security in the transmission and integrity of the data because the information that is sent can be susceptible to capture and modification during the transmission process.

TLS is implemented properly on the client and server side, you make sure that no eavesdropper can intercept your communication and you get that additional layer of security that is important for application-level authentication and authorization.

When configuring an encrypted communication through certificates, it is necessary to verify the characteristics of the equipment, in an unencrypted communication the equipment does not consume many of its resources, but when an encrypted communication is used, the equipment increases the use of its means, making the equipment work slow.

References

1. Nielsen NewsWire, “State of the App-Nation—a Year of Change and Growth in US Smartphones,” May 2012, [Online]. Available: www.nielsen.com.
2. H. C. Hwang, J. Park, and J. G. Shon, “Design and implementation of a reliable message transmission system based on mqtt protocol in IoT,” Wireless Personal Communications, vol. 91, no. 4, pp. 1765–1777,2016. [Online]. Available: http://dx.doi.org/10.1007/s11277-016-3398-2
3. K. Tang, Y. Wang, H. Liu, Y. Sheng, X. Wang, and Z. Wei, “Design and implementation of push notification system based on the mqtt protocol,” in International Conference on Information Science and Computer Applications (ISCA 2013), 2013, pp. 116–119
4. IBM. (2017, March) “Introduction to mqtt”. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSFKSJ_7.5.0/com.ibm.mm.tc.doc/tc00000_.htm
5. K. M. Bell, D. N. Bleau, and J. T. Davey, “Push notification service,”Nov. 22 2011, uS Patent 8,064,896
6. HiveMQ, “Introducing the MQTT Security Fundamentals”. [Online]. Available: http://www.hivemq.com/blog/introducing-the-mqtt-security-fundamentals
7. Barata, D., Louzada, G., Carreiro, A., & Damasceno, A. (2013). System of acquisition, transmission, storage and visualization of Pulse Oximeter and ECG data using Android and MQTT. Procedia Technology, 9, 1265-1272
8. Pan, Z., Liang, X., Zhou, Y. C., Ge, Y., & Zhao, G. T. (2015, June). Intelligent Push Notification for Converged Mobile Computing and Internet of Things. In *Web Services (ICWS), 2015 IEEE International Conference on* (pp. 655-662). IEEE.
9. Google, “Firebase Cloud Messaging (FCM) for Android, Available:”https://firebase.google.com/docs/cloud-messaging/l”, 2017, [Online; accessed 24 March-2017].
10. Apple, “Apple Push Notification Service”, Available:”https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html#//apple_ref/doc/uid/TP40008194-CH8-SW1, 2016, [Online; accessed 24-March-2017].
11. Microsoft, “Push notifications for Windows Phone 8”, Available:”https://msdn.microsoft.com/en-us/library/windows/apps/ff402558(v=vs.105).aspx,2016, [Online; accessed 24-March-2017].
12. Amazon, “Amazon Device Messaging API,” https://developer.amazon.com/device-messaging, 2017, [Online; accessed 24-March-2017].
13. Facebook, “Push Notifications”, Available:”https://developers.facebook.com/blog/post/2016/04/12/facebook-analytics-for-apps-adds-push-notifications-deeper-insights/”, 2017, [Online; accessed 24-March-2017].

14. Singh, M., Rajan, M. A., Shivraj, V. L., & Balamuralidhar, P. (2015, April). Secure mqtt for internet of things (IoT). In *Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on* (pp. 746-751). IEEE.
15. IBM MQTT Protocol Specification. Access Dec 22, 2015, http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html.
16. Lee, S., Jung, I., Kim, H., & Ju, H. (2013). The design of integrated mobile SNS gateway structure. In 2013 15th Asia-Pacific network operations and management symposium (APNOMS) (pp. 1–3).
17. Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008, January). MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on* (pp. 791-798). IEEE.
18. Gero Mühl. Large scale content based publish subscribe systems: [PhD Thesis]. Darmstadt University of Technology, 2002.
19. HiveMQ, "Approach to security in MQTT", 2017, Available: "http://www.hivemq.com/blog/introducing-the-mqtt-security-fundamentals" [Online; accessed 25-March-2017].
20. A. Inc. "Apple developer". [Online]. Available: https://developer.apple.com/develop/
21. Adolfo Martinelli, "Swift MQTT", 2016, [Online]Available: https://github.com/aciidb0mb3r/SwiftMQTT
22. Lem, "Android MQTT", 2016, Available: [Online] https://github.com/bytehala/android-mqtt-quickstart.