REVISTA
ODIGOS

Período febrero - mayo 2022
Vol. 3, Núm. 1
rodigos@uisrael.edu.ec
https://doi.org/10.35290/ro.v3n1.2022.539 • e-ISSN: 2697-3405

Android service to interface mosquitto messaging broker (MQTT)

Fecha de recepción: 2021-12-26 • Fecha de aceptación: 2022-01-05 • Fecha de publicación: 2022-02-10

Cristian Mauricio Gallardo Paredes[1]
Universidad Politécnica de Tomsk, Rusia
kristianmaurisio1@tpu.ru
https://orcid.org/0000-0002-5361-9144

Patricia del Rocío Rodríguez Fiallos[2]
Ministerio de Educación Distrito 18D02, Ecuador
patriciad.rodriguez@educacion.gob.ec
https://orcid.org/0000-0002-5213-1008

Francisco Javier Galora Silva[3]
Universidad Internacional de la Rioja, Ecuador
francisco.galora049@comunidadunir.net
https://orcid.org/0000-0002-5464-5336

ABSTRACT

Real-time messages are used at a high level by different computer applications, there are different servers, protocols, etc. that can be used and are available on the Internet, however, these are created in such a way that the consumption of computational resources is very high and would only be optimal if we use a server with great features. This research work aims to develop a service on Android service to interface mosquitto messaging broker (MQTT) and provide a suitable mechanism to receive requests from an Android application at the same time also notify that some data has arrived, data are stored in the SQLite manager, which uses few computational resources and is suitable for small applications, which are testing protocols and messaging servers. The experimental results show that the service meets the initial objectives of the work, allowing to create of a messaging system based on publishers and subscribers for the Android platform.

KEYWORDS: service, android, Mosquito, MQTT

RESUMEN

Los mensajes en tiempo real son utilizados en un alto nivel por diferentes aplicaciones informáticas, existen diferentes servidores, protocolos, etc., que se pueden utilizar y están disponibles en Internet; sin embargo, estos están creados de tal manera que el consumo de recursos computacionales es muy elevado y solo sería óptimo si se utiliza un servidor con grandes prestaciones. Este trabajo de investigación tiene como objetivo desarrollar un servicio sobre Android, que sirva de interfaz con el broker de mensajería Mosquitto (MQTT) que proporcione un mecanismo adecuado para recibir peticiones de una aplicación Android y al mismo tiempo también notifique que han llegado unos datos, mismo que se almacenan en el gestor SQLITE, que utiliza pocos recursos computacionales y es adecuado para pequeñas aplicaciones, que están probando protocolos y servidores de mensajería. Los resultados experimentales muestran que el servicio cumple con los objetivos iniciales del trabajo, permitiendo crear un sistema de mensajería basado en editores y suscriptores para la plataforma Android.

PALABRAS CLAVE: servicio, Android, Mosquito, MQTT

Introduction

With the notoriety of cell phones, individuals are progressively subject to their cell phones to receive continuous messages, so texting is especially significant (Zhang et al., 2007). As of now, we can sidestep the transporters, through a standard TCP/IP organization to send messages straightforwardly to the cell phone (Peiji & Yanai, 2001).

It's obviously true that over the most recent couple of years, cell phones have altered the entire world, and a piece of this significant world pattern has involved changing the manner in which individuals speak with each other; one of the components of this current has been texting administration. The push message administrations were executed different convention structures like XMPP, CoAP, and MQTT. These different conventions are utilized for each unique circumstance, specifically the MQTT convention was intended to chip away at low-power gadgets pleasantly as a lightweight convention and has been utilized in numerous IoT gadgets and texting frameworks (Hwang et al., 2016).

MQTT protocol become our best option due for its undeniable potential benefits. It is an open, basic, lightweight, and simple to-carry out informing convention. At first, it was intended to associate huge quantities of far-off sensors and control gadgets (Silva et al., 2017). The convention has been applied in an assortment of inserted frameworks. Clinics utilize this convention to speak with pacemakers and other clinical gear providers. Oil and gas organizations use it to screen oil pipelines large number of miles away (Shaoyue et al., 2012).

A portable application that utilizes MQTT sends and gets messages by calling a MQTT library. Messages are traded through a MQTT informing server. The MQTT customer and server handle the intricacies of conveying messages dependably to the versatile application and monitor the cost of organization the board.

MQTT applications run on cell phones, for example, cell phones and tablets. MQTT is likewise utilized in telemetry to get information from sensors and to control them from a distance. For cell phones and sensors, MQTT offers a profoundly adaptable distribution/membership convention with secure conveyance.

A few review papers are tending to various attributes of distribute/buy in based frameworks, like plan, execution, quality necessities, directing calculations, and so forth In "Nature of Administration in Wide-Scale Distribute Buy in Frameworks", the creators research the cutting edge modern and scholarly distribute/buy in arrangements zeroing in on adaptability and quality prerequisites, while in "The Many Essences of Distribute/Buy in", the creators present order dependent on correspondence, different distribute/buy in plans, plan, and executions, yet as far as we could possibly know, there is no overview which recognizes key-necessities for IoT distribute/buy in arrangements.

The following research is also carried out in this important area, such as:

A. Design and implementation of a reliable message transmission system based on

MQTT protocol in IoT

This paper planned and executed a dependable message transmission framework utilizing MQTT convention to keep up with requesting between messages for the workplace. This (framework) comprises of MQTT convention, solid message transmission server, and customer module (Hwang et al., 2016).

B. Design and implementation of push notification system based on the MQTT protocol

This paper depicted a technique for pushing warning framework dependent on the MQTT convention. It tends to be utilized to tackle the issue of moment pushing different messages from the server to the versatile customer (Tang et al., 2013).

C. System of acquisition, transmission, storage and visualization of pulse oximeter and ECG data using android and MQTT

The creators present an arrangement of wellbeing information assortment, transmission, and capacity intended for electrocardiography (ECG) and Heartbeat Oximetry results, the objective is to address this test by making a framework for gaining and communicating information through Bluetooth to an Android versatile stage, which sends it to a distant server, where the information is put away in a data set and opens up for perception. This empowers any far off client to get to imperative information on a patient without being actually present (Barata et al., 20139.

D. Correlation analysis of MQTT loss and delay according to QoS level

This examination breaks down the MQTT message transmission process which comprises of truly wired/remote distribute customer, specialist server, and buy in customer. By communicating messages through 3 degrees of QoS with different sizes of payloads, we have caught parcels to examine start to finish deferrals and message misfortune (Lee et al., 2013).

All these papers are of great help as a base for the realization of this project, because they allow understanding the benefits that can be obtained when using mosquito MQTT and its linkage with other technologies that would greatly benefit people, this can be evidenced by the investigations carried out in this important area.

The purpose of this research is to perform an android service to interface Mosquitto Message Broker (MQTT), offering a suitable mechanism to receive requests from Android applications and also to notify that some data has arrived.

Methodology

In this chapter, we propose a method for sending and receiving messages, this is done through an android service, which allows you to create the topic, make subscriptions, perform publication, display all messages. For the experiment, we used Linux-based Debian 8 for the server and an open-source project Mosquitto MQTT for Broker server software.

It is very important to keep the guarantee of sending messages reliably, the service developed in Android through the mobile application communicates with other IoT devices messages must have atomicity, consistency, and performance (Schiper & Raynal, 1996). In addition, the mobile application for reliable message transmission should be able to view the previous messages for reliable message transmission or communication history.

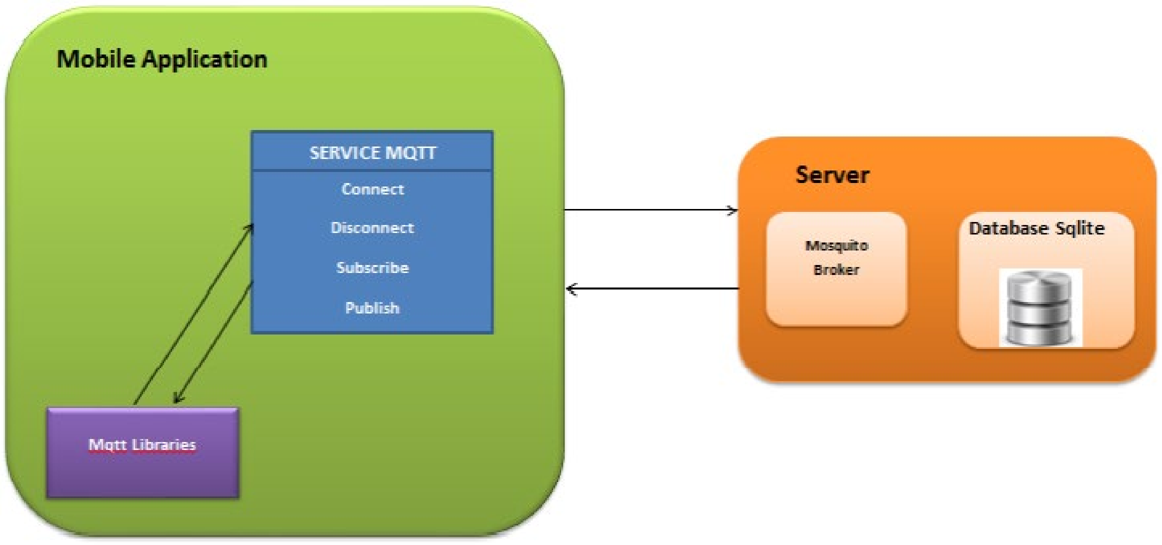
The definition of all components of both software and hardware the existing relationships between them are the most relevant aspects at the time of execution of a project. Next, all the architecture adjacent to this project will be described and specified.

2.1 MQTT service architecture

The service is within the mobile application, the service uses MQTT libraries to perform the functions of: Connect, Disconnect Publish, subscribe, QoS. By executing a .sh file the messages are stored in an SQLite database (see *Figure 1*).

Figure 1.

Service Architecture



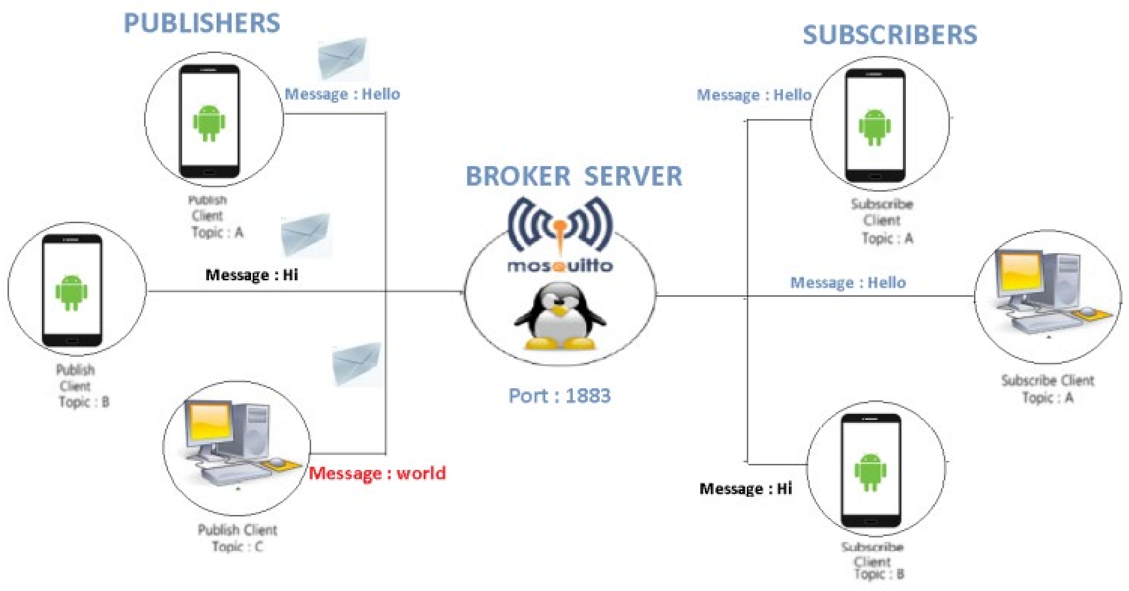
The central node that goes about as a server or representative, equipped for working up to 500000 messages. It is the merchant definitively the component responsible for dealing with the organization and sending the messages. Correspondence dependent on themes, the customer distributes the message, makes and the hubs that wish to get it should prefer it. The intermediary server associates with a SQLite information base utilizing a clump cycle to store the messages coming from the Distributers, MQTT customers preferred a point.

2.2 Architecture of MQTT message transmission between multiple clients

Customers under remote organization climate were tried under Android climate. The remote climate can be viewed as reasonable, since the correspondence from portable climate goes through 3G organization and arrives at specialist waiter (see *Figure 2*).

Figure 2.

The general architecture of MQTT message transmission between multiple clients



It investigates message misfortune and start to finish delay by gathering bundles among customer and server during the 5 minutes' estimation. With regards to start to finish delay, we use timestamps shaped as bundles move from the membership server to distribute the customer through the representative server. The retransmission demand bundles were counted when of five minutes to gauge message misfortune.

2.3 Implementation

In the first place, performed standard tests to comprehend the MQTT standards with online test agents (test.mosquitto.org, broker.mqttdashboard.com) and the order line: preferring points and distributing messages in those subjects. From that point onward, research was done to assess what was at that point made with respect to Java.

MQTT customers to characterize a beginning stage. We ran over various programming like Java customer Application Programming Points of interaction (APIs) and dealers, which are recorded in *Table 1*.

Table 1.

Implementation features

Brokers	License	Java Client APIS
Active MQ	Open-source	Eclipse Paho - a Java client developed by the Eclipse Foundation (Iyer et al., 2018)
Hive MQ	Commercial	MQTT-client - a Fuse source Java MQTT client with a variety of API styles (Manh Pham et al., 2019)
IBM WebSphere Message broker	Commercial	IBM WebSphere MQ Telemetry provides a Java client API (Katsikeas et al., 2017)
Mosquito	Open Source	Free

2.3.1 Local Server MQTT

First import the repository package signing key:

- wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
- sudo apt-key add mosquitto-repo.gpg.key

Then make the repository available to apt:

- cd /etc/apt/sources.list.d/

Then one of the following, depending on which version of Debian

- sudo wget http://repo.mosquitto.org/debian/mosquitto-wheezy.listsudo
- wget http://repo.mosquitto.org/debian/mosquitto-jessie.list

Update and install mosquitto

- apt-get update 2. apt-get install mosquitto

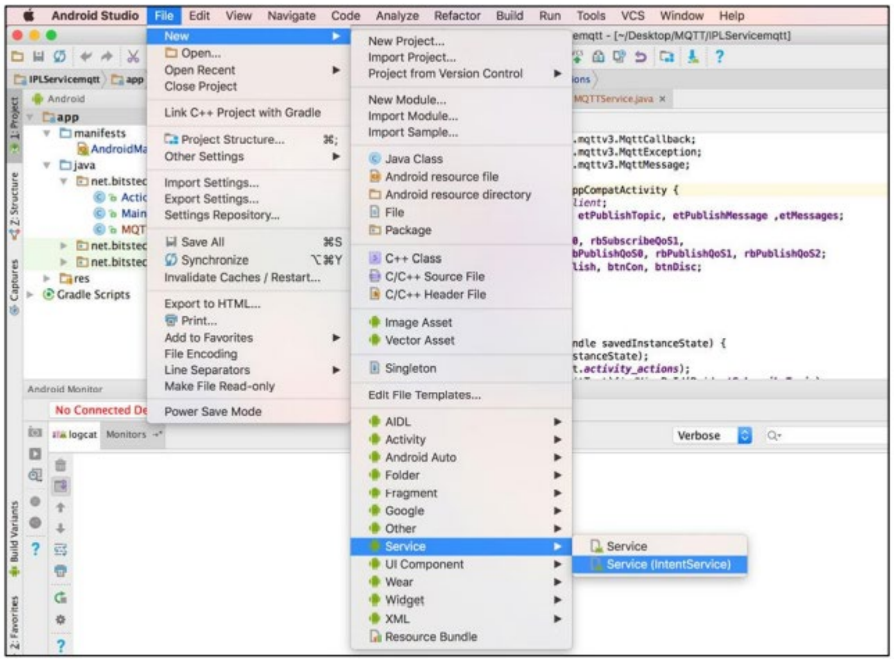
Run mosquito and see the status

- Mosquitto –c /etc/mosquitto/mosquitto.conf 2. Service mosquitto status

Create service on android studio, see *Figure 3.*

Figure 3.

Service on Android Studio



2.3.2 Message storage SQLite

Before configuring the MQTT server you need a requirement to have installed an Ubuntu or Debian server, with a non-root, sudo-enabled user and basic firewall.

Installing SQLite (Nemetz et al., 2018)

- sudo apt-get update;
- sudo apt-get install sqlite3 libsqlite3-de

Create a database, if you are still in the sqlite3 program, exit with. Quit in the SQLite flag, then run the command:

- sqlite3 $HOME/MESSAGE.DB

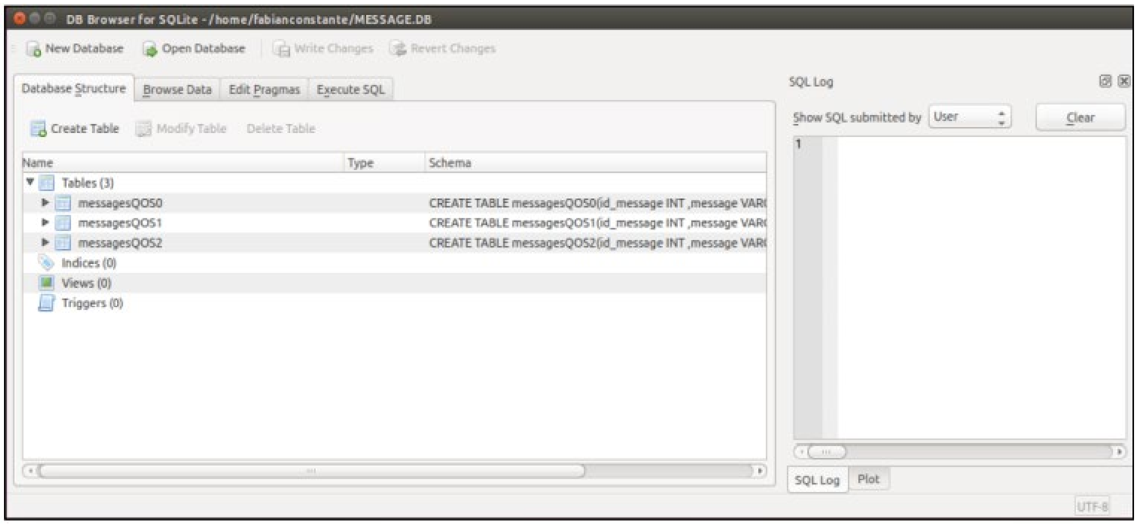
Creating tables corresponding to each QoS (see *Figure 4).*

- sqlite3 $HOME/MESSAGE.DB “CREATE TABLE messagesQOS0(id_message INT ,message VARCHAR(200), topic VARCHAR(50),date text);”;
- sqlite3 $HOME/MESSAGE.DB “CREATE TABLE messagesQOS1(id_message INT ,message VARCHAR(200), topic VARCHAR(50),date text);”;

- sqlite3 $HOME/MESSAGE.DB “CREATE TABLE messagesQOS2(id_message INT ,message VARCHAR(200), topic VARCHAR(50),date text);”;

Figure 4.

Message database



Name	Type	Schema
Tables (3)		
messagesQOS0		CREATE TABLE messagesQOS0(id_message INT ,message VAR
messagesQOS1		CREATE TABLE messagesQOS1(id_message INT ,message VAR
messagesQOS2		CREATE TABLE messagesQOS2(id_message INT ,message VAR
Indices (0)		
Views (0)		
Triggers (0)		

Results

3.1 Functional testing

Functional testing is a quality confirmation (QA) process and a kind of discovery testing that puts together its experiments with respect to the details of the product part under test. Capacities are tried by taking care of them input and inspecting the result, and inward program structure is seldom thought of (dissimilar to white-box testing). *Table 2,* portrays exhaustively the particular of the tests that are performed for usefulness

Table 2.

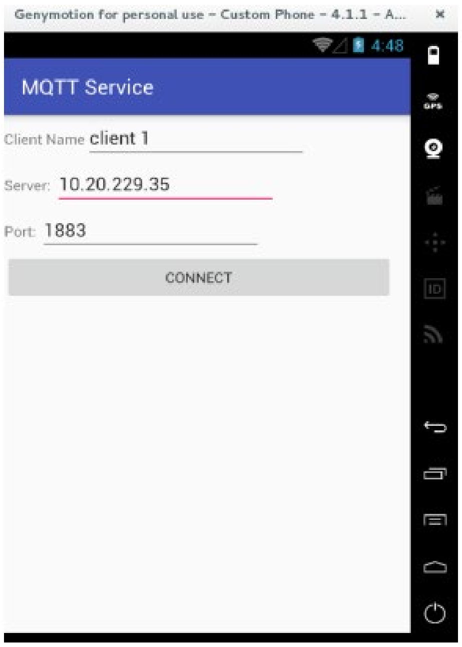
Usefulness test

Case	Event number	Test	Result
1. Connecting the Client to the MQTT Server	1	The client specifies the IP address of the host message broker and the port number assigned by MQTT to verify the status of the connection.	The connection to the MQTT server is successful.
	2	Display the successful connection message to the server	The displayed message indicates that the connection is successful.
2. Subscription to a topic and QoS	1	In the Android app, the subscription parameter and the quality parameter are sent, to press the "Subscribe"	Successful subscription to a topic
	2	Display message indicating subscription to the topic	The message indicates successful subscription
3. Message publishing and QoS	1	Enter the channel where the user has subscribed, write any message and press the "Publish" button, verify that the published message has been sent to the server.	The MQTT server displays the received message
	2	From the server send a reply message indicating that the message has been successfully published	The message indicates that data has been successfully published
4. Storing sent messages in a database	1	All messages sent in each QoS from different clients will be stored	Qo0 and Qo1 May have lost messages Qo2 This level has no loss in sent messages

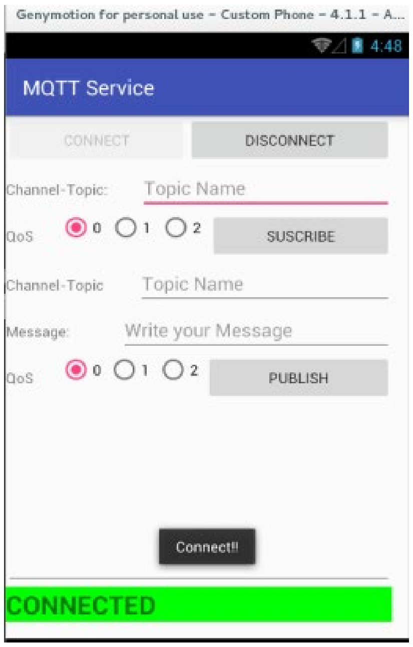
1. **Connecting the client to the MQTT Server.** The connection to the server from the application through the service was satisfactory (see *Figure 5*).

Figure 5.

Connecting the Client to the MQTT Server



Client Name client 1
Server: 10.20.229.35
Port: 1883



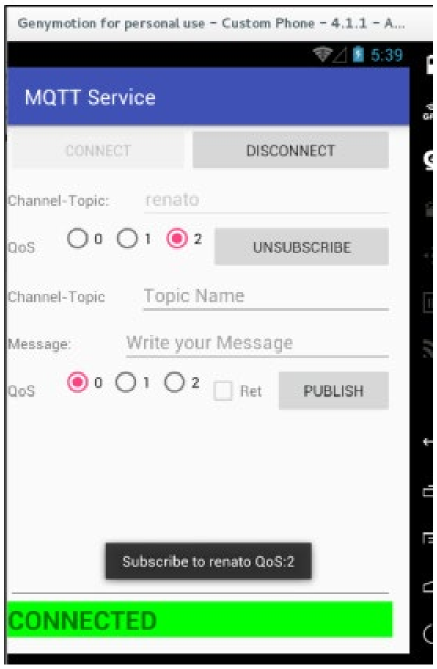
Connect!!

CONNECTED

2. **Subscription to a topic and QoS. -** The subscription to a topic was satisfactory, this is sow in *Figure 6*.

Figure 6.

Subscription Interface to the topic



Channel-Topic: renato
QoS [✓ 2]

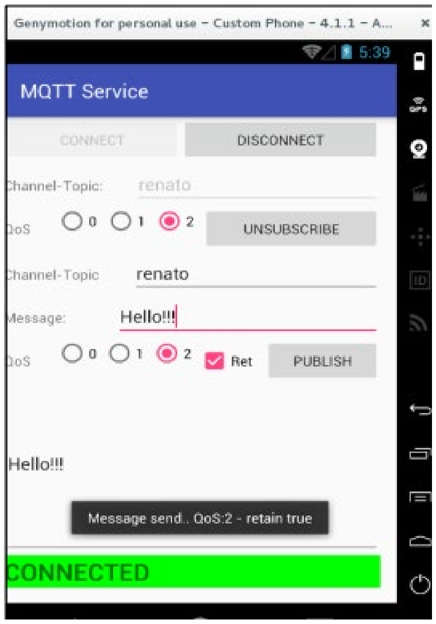
Subscribe to renato QoS:2

CONNECTED

3. **Message publishing and QoS.** Messages publishing on a topic was satisfactory, see *Figure 7*.

Figure 7.

Publication of an MQTT message



Channel-Topic: renato
QoS ○ 0 ○ 1 ● 2
Channel-Topic renato
Message: Hello!!!
QoS ○ 0 ○ 1 ● 2 ✓ Ret

Hello!!!

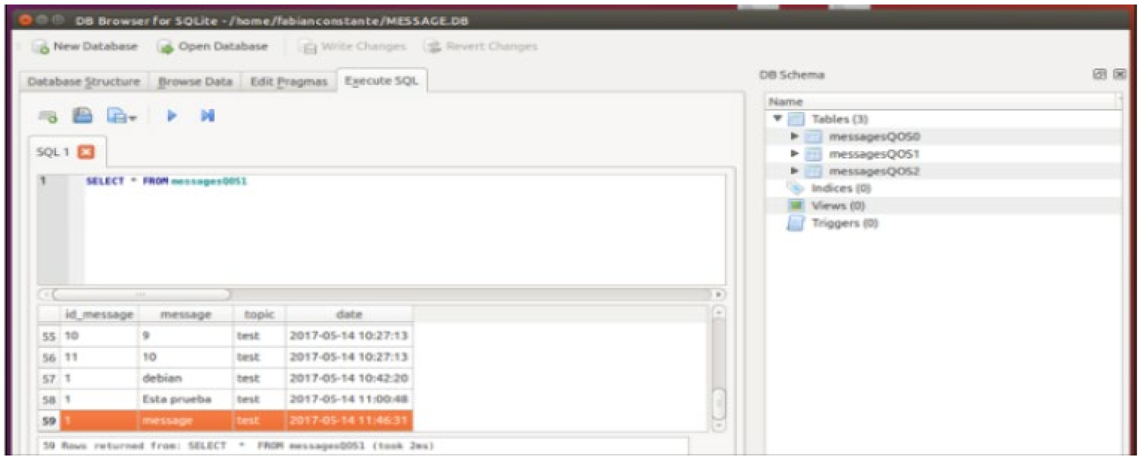
Message send.. QoS:2 - retain true

CONNECTED

4. **Storing sent messages in a database**. - The user submits, the message is automatically stored in a SQLite Database, this is possible when the .sh program is running. As shown in *Figure 8*.

Figure 8.

Storage of messages sent through the MQTT Service



```
SELECT * FROM messagesQOS1
```

	id_message	message	topic	date
55	10	9	test	2017-05-14 10:27:13
56	11	10	test	2017-05-14 10:27:13
57	1	debian	test	2017-05-14 10:42:20
58	1	Esta prueba	test	2017-05-14 11:00:48
59	1	message	test	2017-05-14 11:46:31

```
59 Rows returned from: SELECT * FROM messagesQOS1 (took 2ms)
```

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.

We specified the design of a pushing notification system and discussed details of key techniques that make the system effective and easy to maintain. This message push system is based on MQTT protocol and makes it possible to push messages to clients in real-time.

From the functionality tests, positive results were obtained, users connected and sent messages correctly and quickly, this on a local server. While in the performance tests it was verified that in QoS1 a message was lost and the reception of the messages with the levels QoS0 and QoS2 were successful. This ensures that messages arrive correctly to users.

References

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.

Hwang, H. C., Park, J., & Shon, J. G. (2016). Design and implementation of a reliable message transmission system based on MQTT protocol in IoT. *Wireless Personal Communications, 91*(4), 1765-1777.

Iyer, S., Bansod, G. v., Praveen Naidu, V., & Garg, S. (2018). Implementation and Evaluation of Lightweight Ciphers in MQTT Environment. 3rd International Conference on Electrical, Electronics, Communication, *Computer Technologies, and Optimization Techniques, ICEEECCOT 2018,* 276–281. https://doi.org/10.1109/ICEEECCOT43722.2018.9001599

Katsikeas, S., Fysarakis, K., Miaoudakis, A., Van Bemten, A., Askoxylakis, I., Papaefstathiou, I., & Plemenos, A. (2017). Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol. In *2017 IEEE Symposium on Computers and Communications (ISCC)* (pp. 1193-1200).

https://doi.org/10.1109/ISCC.2017.8024687

Lee, S., Kim, H., Hong, D. K., & Ju, H. (2013). Correlation analysis of MQTT loss and delay according to QoS level. In *The International Conference on Information Networking 2013 (ICOIN)* (pp. 714-717).

Manh Pham, L., Nguyen, T.-T., & Tran, M.-D. (2019). A Benchmarking Tool for Elastic MQTT Brokers in IoT Applications. *International Journal of Information and Communication Sciences, 4*(4), 59. https://doi.org/10.11648/J.IJICS.20190404.11

Nemetz, S., Schmitt, S., & Freiling, F. (2018). A standardized corpus for SQLite database forensics. *Digital Investigation, 24*, S121–S130. https://doi.org/10.1016/J.DIIN.2018.01.015

Peiji, L, & Yanai, W. (2001). The Application of Push Technology in Mobile Internet *Communications World, 31*, pp. 31 - 32.

Schiper, A., & Raynal, M. (1996). From group communication to transactions in distributed systems. *Communications of the ACM*, *39*(4), 84-87. https://dl.acm.org/doi/abs/10.1145/227210.227230

Shaoyue, H., Xiaodong, X., & Zuyuan, M. (2012). The Application of Active Push Technology in Mobile Collaboration Education [J]. *Modern Education Technology*, *4*, 100-103.

Silva, C., Toasa, R., Martinez, H. D., Veloz, J., & Gallardo, C. (2017). Secure push notification service based on MQTT protocol for mobile platforms. In *XII Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento y Congreso Ecuatoriano en Ingeniería de Software* (pp. 69-84).

Tang, K., Wang, Y., Liu, H., Sheng, Y., Wang, X., & Wei, Z. (2013). Design and implementation of push notification system based on the MQTT protocol. In *International Conference on Information Science and Computer Applications (ISCA 2013)* (pp. 116-119). Atlantis Press.

Zhang, W. M., Zhang, M., Bi, J., & Qin, Z. (2007). Instant messaging: The present and the future. *MINIMICRO SYSTEMS-SHENYANG-*, *28*(7), 1162-1168. https://en.cnki.com.cn/Article_en/CJFDTotal-XXWX200707001.htm