MQTT for IoT-based Applications in Smart Cities *

Dr. Samer Jaloudi **

* **Received: 6/1/2018, Accepted: 24/3/2018.** **DOI**: https://doi.org/10.5281/zenodo.2582892

** **Assistant Professor/Al-Quds Open University/Palestine.**

Abstract

This research presents a study on the use of the MQTT communications protocol for the Internet of Things in Smart City applications. A network model is proposed and a typical practical scenario is developed based on MQTT protocol, that cope with the requirements of some Smart City applications, mainly those which use event-based messages. Many market-available embedded electronic systems were employed for this scenario including the inexpensive Wi-Fi platform ESP8266, Arduino, Raspberry, in addition to some sensors and actuators. The network model is chosen based on the TCP/IP model, and the application layer is totally depending on the MQTT protocol that employs JavaScript Object Notation (JSON) to solve the problem of interoperability. To evaluate the protocol for small-to-medium, IoT-based, business applications of Smart Cities, some available free Open Source Software (OSS) of MQTT servers and clients were compared and tested against latency over the cloud. The protocol shows good results for cloud-based, small-to-medium business applications that depend on event-based message-oriented communication paradigms. Since the protocol defines three levels of quality of service (QoS), the simulations and the tests were conducted for QoS type zero (QoS0) to get the best results.

Keywords— CoAP, HTTP, Internet of Things, JSON, Latency, MQTT, Network Model, Publish-and-Subscribe, One-to-Many, Smart City Applications.

توظيف بروتوكول الاتصالات MQTT

في التطبيقات المستندة إلى إنترنت الأشياء

في المدن الذكية

ملخص:

تقدم هذه الورقة البحثية دراسة حول توظيف بروتوكول الاتصالات المسمى MQTT لإنترنت الأشياء Internet of Things في تطبيقات المدن الذكية. تم اقتراح نموذج شبكة Network Model وتطوير سيناريو عملي نموذجي اعتمادا على بروتوكول الاتصالات MQTT، والذي يراعي احتياجات بعض تطبيقات المدن الذكية خصوصا تلك التي تستخدم رسائل قائمة على الحدث. تم توظيف العديد من النظم الإلكترونية المتكاملة لهذا السيناريو وتشمل النظام غير المكلف ESP8266، منصة الأردوينو، منصة الراسبري، إضافة إلى بعض المجسات والمشغلات. إضافة إلى اختيار نموذج الشبكة بالاعتماد على النموذج المعياري TCP/IP، وطبقة التطبيقات في النموذج تعتمد بشكل كلي على بروتوكول MQTT والذي يوظف رموز أهداف جافا سكريبت JSON لحل مشكلة التوافقية Interoperability. من أجل تقييم البروتوكول للأعمال الصغيرة–إلى–المتوسطة، التي تعتمد انترنت الأشياء في تطبيقات أعمال المدن الذكية، فقد تم عمل مقارنة لبعض برمجيات MQTT المتوفرة للزبون client والخادم server من نوع مفتوح المصدر OSS وإجراء فحص لها وتحديدا فحص Latency لبعض الخوادم على شبكة الانترنت. أظهر البروتوكول نتائج جيدة لتطبيقات الأعمال الصغيرة–إلى–المتوسطة التي توظف خوادم على الإنترنت، والتي تعتمد على الاتصال ذو الرسائل القائمة على الحدث event-based message-oriented communications. بما أن البروتوكول يعرف ثلاثة مستويات من نوعية الخدمة QoS، فقد تم إجراء المحاكاة والفحص على المستوى الأول من نوعية الخدمة QoS0، وذلك من أجل الحصول على أفضل نتائج البروتوكول.

كلمات مفتاحية: CoAP، HTTP، انترنت الأشياء، JSON، السكون، MQTT، نموذج شبكة، نشر–و–اشتراك، واحد–إلى–عديد، تطبيقات المدينة الذكية.

Introduction

Smart City applications such as smart transportation, smart healthcare, smart buildings, smart homes and smart meters require the use of standard telecommunication protocols and infrastructures (Jaloudi, 2015). Internet of Things (IoT) introduces its infrastructure, which is the Internet or Intranet, as an inexpensive and available telecommunication infrastructure for Smart City applications. Information integration, in such applications, could be realized via Internet-based standards. Many standard communication protocols have been proposed for IoT in Smart City applications, including but not limited to, Message Queuing Telemetry Transport (MQTT) (OASIS, 2014 and ISO/IEC, 2016), Constrained Application Protocol (CoAP) (Shelby et. al,

2014), and Hyper Text Transfer Protocol (HTTP) (Fielding and Reschke, 2014).

The MQTT version 3.1 is a telecommunication protocol released by IBM in 2010 (Locke, 2010). The new version 3.1.1, became an OASIS standard on October 2014 (OASIS, 2014) and an ISO/IEC standard on 2016 (ISO/IEC, 2016). The MQTT is publish-subscribe messaging transport protocol, lightweight, open standard, simple to implement programmatically and loosely couples clients to the server in asynchronous communication mechanism. These features make it a good choice for IoT-based applications and machine-to-machine (M2M) communications (Locke, 2010). However, other IoT protocols, namely, CoAP and HTTP are both request-response protocols, and tightly-couple clients to servers in synchronous communication mechanism.

This research paper is concerned with the evaluation of MQTT protocol for IoT-based Smart City applications such as smart homes, smart lighting, smart healthcare, etc. A network model is proposed based on MQTT over TCP/IP model as the basis of a practical scenario for Smart City applications. For seamless information integration and interoperability, the user data is formatted in JavaScript Object Notation (JSON) (Bray, 2014). The importance of this study originates from the idea of using the MQTT protocol for small-to-medium business, IoT-based, Smart City sectors. Thus, the protocol is simulated against latency, for many clients exchanging messages via online-based MQTT servers, available free-of-charge for testing purposes. In addition, the protocol is simulated, against losses, on a local machine using a commercial MQTT server. The author concluded that the MQTT protocol is suitable for small-to-medium business, IoT-based applications that exchange messages with online-based MQTT servers, and for medium-to-big, IoT-based applications that exchange messages with local MQTT servers.

The rest of the paper is organized as follows: The first section introduces a literature review; latency effects were measured for online and local MQTT servers in the second section. A network model is proposed in the third section; followed by a development of practical scenario in the fourth section. The fifth section concludes the paper.

Literature Review

Many research papers investigated the IoT protocols for Smart City sectors. For example, authors in (Kodali and Soratkal, 2016) have developed a home automation system based on MQTT. A room temperature and fire alarm/suppression IoT service using MQTT on Amazon web service is built in (Kang et al., 2017). In (Yi et al., 2016), a mobile health monitoring system is designed and implemented based on MQTT. A scalable tracking system for public buses based on MQTT is developed by the authors of (Lohokare et. al, 2017), and a smart bus for smart city in (Sharad et. al, 2017). However, a remote health monitoring system for smart regions is built in (Khoi et al, 2015) based on CoAP protocol, and an interoperable messaging system for IoT healthcare services is implemented by (Oryema et. al, 2017).

A performance comparison between HTTP and MQTT is made by (Yokotani and Sasaki, 2016) on required network resources for IoT. In addition, performance analysis of MQTT, HTTP, and CoAP is estimated in (Joshi et. al, 2017) for IoT based monitoring of smart home. Authors in (Thota and Kim, 2016) discussed and analyzed the efficiency, usage and requirements of MQTT and CoAP.

MQTT Messaging Technology

The main features of MQTT include, from different communication aspects, infrastructure, architecture, mechanism, model, messaging pattern, methodology and transmission paradigm. The protocol uses the client-server communication architecture, based on the publish-subscribe model, which is message-oriented protocol. Therefore, MQTT is event-based, one-to-many protocol. In addition, it uses the inexpensive and available communication infrastructure, which is Internet or Intranet in wire mode (Ethernet – IEEE 802.3) or wireless mode (Wi-Fi – IEEE 802.11) that may employ either IPv4 or IPv6 in the network layer. In the transport layer, the MQTT uses TCP port number 1883.

On the other hand, and according to the MQTT protocol specifications (OASIS, 2014), the MQTT packet have a fixed header of two bytes, followed by variable length header depending on topic name length (refer to Figure 2). The fixed

header contains five fields. The first field is a four-bit field and represent the packet type, which is one of the following packet types: CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, PINGREQ, PINGRESP, DISCONNECT. The second field is one-bit, and represents duplicate flag, that must be set to zero for all quality of service (QoS) level 0 messages, because packet duplication is not allowed. The third field represents the QoS level, which takes one of three values; zero (binary 00), one (binary 01), or two (binary 10), and hence, binary value (11) is not allowed. The fourth field is RETAIN. If the RETAIN flag is set to one, the Server must store the complete message and its QoS, so that it can be delivered to future subscribers whose subscriptions match its topic name. The Remaining Length is the fifth field that represents the length of the packet.

For example, the frame format shown in Figure 1 is that of MQTT CONNECT packet. The Options field exists here because the packet contains additional fields. While the PUBLISH packet is similar but with reserved (set to zero) bits seven to four, the CONNACK packet frame contains four bytes, of which two bytes represent the fixed header and another two bytes for the variable header. However, the DISCONNECT packet frame contains two bytes only that represent the fixed header.

Figure 2 illustrates the exchange of data during the CONNECT packet of MQTT. While the first client (publisher) produces a message in four steps, the second client (subscriber) consumes that message in six steps. The publisher sends a connect packet (CONNECT), with username (un) and password (pwd) if required, to the server (broker) trying to establish a TCP connection. The server acknowledges the attempt with (CONNACK) packet, telling the client (publisher) whether the connection is successfully established or not. Then the client publishes the temperature value via a PUBLISH packet, with temp topic and a value of 22.7 degrees. The client ends the publishing event with the server by sending a DISCONNECT packet. Meanwhile, and in addition to the same aforementioned steps, the subscriber must SUBSCRIBE to the same topic (temp) to receive the published messages. The subscription packet is acknowledged with SUBACK packet.

In fact, MQTT protocol has low data overhead, and targets constrained devices and networks. The protocol is publish-subscribe, one-to-many, TCP-based message-oriented protocol. Since TCP is connection-oriented protocol, MQTT is reliable.

The philosophy of MQTT protocol follows the publish-and-subscribe mechanism and hence its applications are different from those of CoAP and HTTP. The publish-and-subscribe mechanism is event-driven, based on topics, and decoupling the clients (publishers and subscribers) in asynchronous communication methodology. Hence, requirements of many applications of Smart Cities are fulfilled by MQTT, mainly those that do not require the sequenced operation of successive tasks, where the first operation's result is the input of the next operation, such as the closed-loop control. However, for most Smart City applications, like home automation and healthcare, it is enough to transfer data between clients using one-to-many communication mechanism via a broker (server), where simplicity and information integration are of high interest. Hence, MQTT excels in such M2M communications.

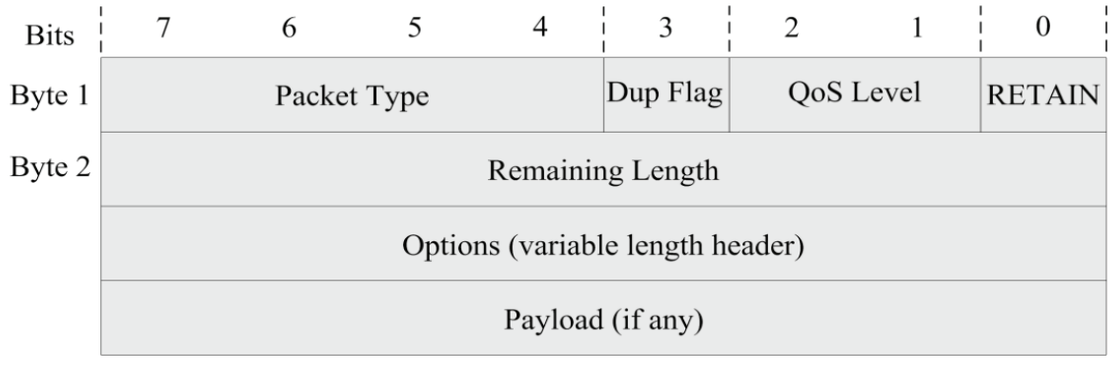

Fig 1

MQTT frame format of CONNECT packet

Latency Measurements

In this section, a survey of MQTT clients and servers is conducted, to emphasize those that are available free-of-charge. Many of these clients and servers were implemented in different programming languages, suitable for a specific operating system or a middleware. Depending upon this survey, free desktop-based and cloud-based servers were compared, tested, and in two cases simulated for latency measurements.

Implementations of MQTT Clients and Servers

Many MQTT servers (brokers) are available, desktop-based, as well as online-based servers (cloud-based brokers). However, those support three levels of QoS, authentication and secured connections are considered. Port 8883 is reserved by IANA for secured MQTT connections over SSL and hence, servers that support such connections were considered. Another aspect is the support of web browsers because MQTT is the inability to pass data to browsers directly. The protocol WebSockets (WS) works as the bridge between browsers and MQTT servers.

Table 1 presents a comparison between available MQTT servers. The comparison is based on many factors such as programming language dependency, whether the server is open source software (OSS) or not, and whether the server supports cloud-based testing or not. Accordingly, the following servers are of high interest: Mosquitto, HiveMQ, Apache Apollo, VerneMQ, HBMQTT, BevyWise, Moquette, ThingStud, Trafero Tstack, and TheThings.

Mosquitto is suitable for testing purposes and for beginners. A free trial of the desktop version of HiveMQ is available for free for six months with limited twenty-five connections, in addition to a cloud-based version on its website, available for testing. Apache Apollo is designed based on the original ActiveMQ. Moquette is available in standalone application and in cloud-based. VerneMQ and HBMQTT support communications over WebSockets protocol locally. BevyWise offers free trial versions of client and server. ThingStud is cloud based and free for non-commercial uses. Trafero/Tstack introduces its hosting in a platform as a service paradigm. TheThings is cloud-based and offers a fifteen-day trial version. These servers were written (implemented) in different programming languages, and therefore the used language must be installed prior to installing the server.

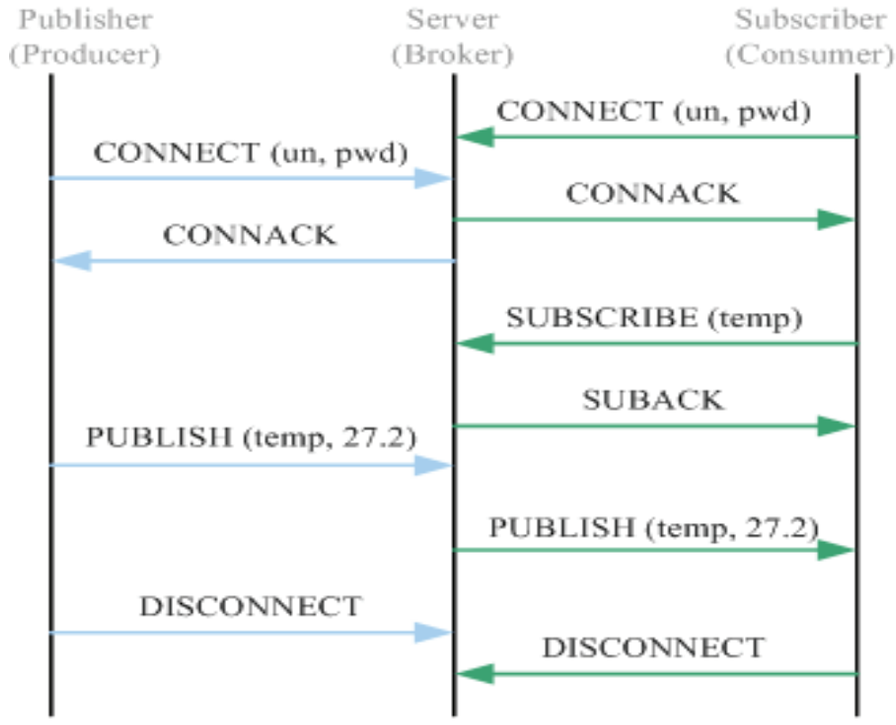

Fig 2

Exchange of messages in MQTT protocol

Some of these online servers (brokers) were tested for the purposes of this paper including the HiveMQ online testing server broker.hivemq.com, the Mosquitto-based web server which is available online as well on test.mosquitto.org, the test server introduced by Eclipse iot.eclipse.org, and that of Moquette broker.moquette.io. The results are shown in tables 2, 3, 4, and 5. The measured latency is conducted for QoS0, in order to get the best results of MQTT. The simulations of clients that connect to those cloud-based servers start counting latency from creating the socket up to closing it including the transmission of messages and network latency. The test is repeated five times for each and then the average is taken, as shown in Table 2, Table 3, Table 4, and Table 5.

Table 1:

Comparison between MQTT servers

Server (Broker)	Website	Cloud-based Testing		OSS	Code
		Address	WS Port		
Mosquitto	mosquitto.org	test.mosquitto.org iot.eclipse.org	80 -	√	C/C++
HiveMQ	www.hivemq.com	broker.mqttdashboard.com broker.hivemq.com	8000 -	X	Java
Apache Apollo	activemq.apache.org/apollo/	X	-	√	Java
VerneMQ	vernemq.com	X	8888	√	Erlang
HBMQTT	hbmqtt.readthedocs.io	X	8080	√	Python
BevyWise	www.bevywise.com	mqttserver.com	8000	X	Python
Moquette	andsel.github.io/moquette/	broker.moquette.io	8080	√	Java
ThingStud	www.thingstud.io	mqtt.thingstud.io	9001	X	JS
Trafero Tstack	hub.docker.com/r/trafero/tstack/	X	-	√	Go
TheThings	www.thethings.com	mqtt.thethings.io	1883	√	JS

Table 2:

Latency of cloud-based server test.mosquitto.org

Test No.	Successive messages	Trial					Average
		1	2	3	4	5	
1	1	261	255	259	256	269	260 ms
2	5	1257	1284	1254	1244	1273	1262 ms
3	25	6227	6264	6228	6246	6282	6257 ms
4	50	17343	17483	19733	17670	17386	17386 ms
5	100	27799	30095	26217	33004	29945	29412 ms

Table 3:

Latency of cloud-based server broker.hivemq.com

Test No.	Successive messages	Trial					Average
		1	2	3	4	5	
1	1	201	195	249	242	231	224 ms
2	5	1148	1025	1157	1088	942	1072 ms
3	25	5343	5902	5458	5347	5423	5495 ms
4	50	9907	9371	9576	9797	9567	9644 ms
5	100	19400	19120	19403	18896	18970	19158 ms

Table 4:

Latency of cloud-based server iot.eclipse.org

Test No.	Successive messages	Trial					Average
		1	2	3	4	5	
1	1	478	490	489	485	481	485 ms
2	5	2372	2369	2370	2379	2371	2372 ms
3	25	13995	13964	13981	14003	14002	13989 ms
4	50	29338	29291	29301	29293	29371	29319 ms
5	100	62999	62895	62504	62249	62786	62687 ms

Table 5:

Latency of cloud-based server broker.moquette.io

Test No.	Successive messages	Trial					Average
		1	2	3	4	5	
1	1	376	367	369	372	375	372 ms
2	5	1766	1766	1761	1788	1782	1773 ms
3	25	10863	10943	10963	10858	10921	10910 ms
4	50	23087	23575	23060	23143	23657	23304 ms
5	100	35510	35287	37285	37219	37267	36514 ms

```
C:\>ping -n 5 -l 37 test.mosquitto.org

Pinging test.mosquitto.org [37.187.106.16] with 37 bytes of data:

Reply from 37.187.106.16: bytes=37 time=106ms TTL=46
Reply from 37.187.106.16: bytes=37 time=104ms TTL=46
Reply from 37.187.106.16: bytes=37 time=103ms TTL=46
Reply from 37.187.106.16: bytes=37 time=104ms TTL=46
Reply from 37.187.106.16: bytes=37 time=104ms TTL=46

Ping statistics for 37.187.106.16:
    Packets: Sent = 5, Received = 5, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 103ms, Maximum = 106ms, Average = 104ms
```

Fig 3

Result of the ping command

As a comparison, the ping command is used to ping the hostname test.mosquitto.org, and the result is shown in Figure 3. The -n option specifies the ICMP echo requests, which are five packets instead of the default of four, and the -l option sets the packet size for each request to thirty-seven bytes instead of the default of thirty-two bytes.

Figure 4 illustrates the results of tables 2, 3, 4, and 5, where no lost messages were registered. The server broker.hivemq.com has the lowest latency of them all. In fact, these measurements do not depend only on network latency, but also on the server itself.

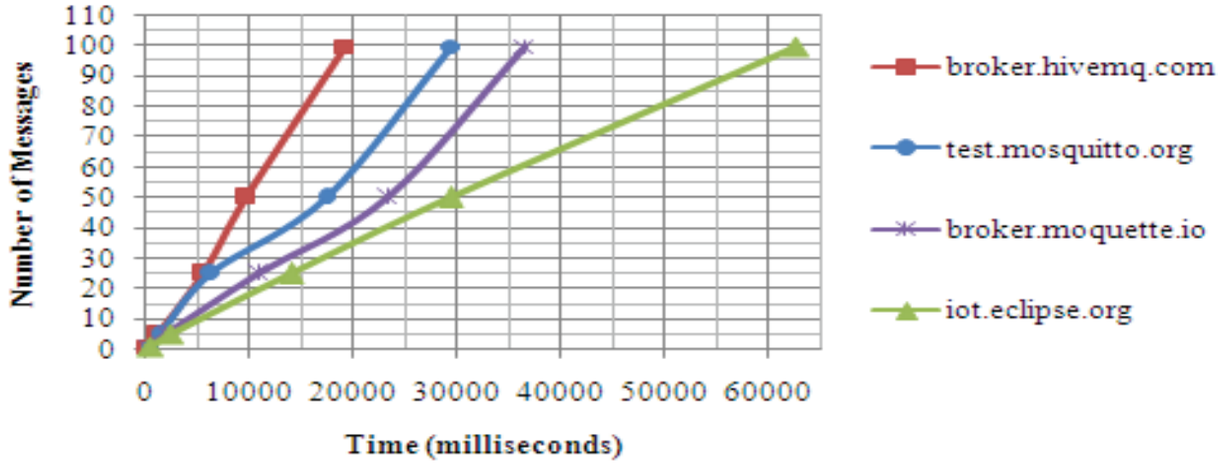

Fig 4
Latency of native MQTT messages over cloud-based test servers

Table 6:
Tests of commercial MQTT server against losses

Test No.	Number of Clients	Messages per Client	Total Messages	Losses
1	1	100	100	7
2	2	50	100	22
3	5	20	100	32
4	10	10	100	24
5	20	5	100	18

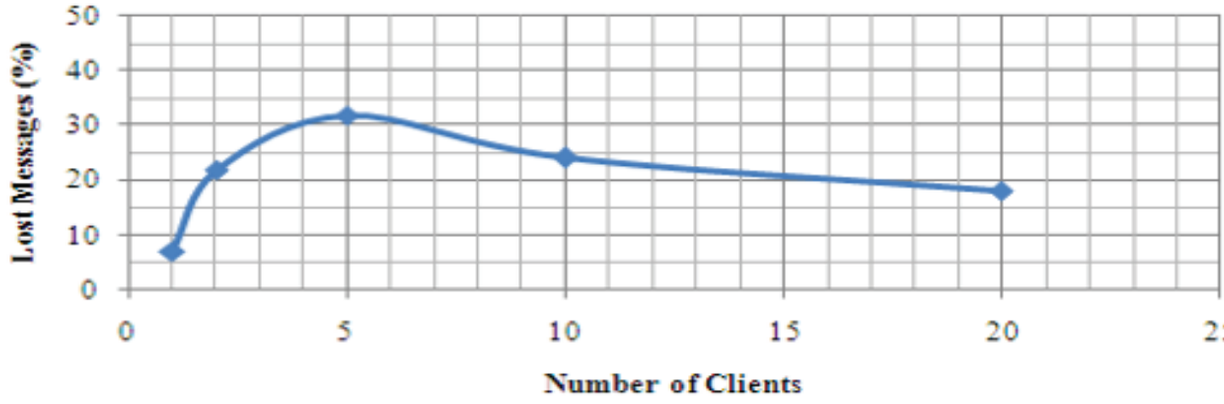

Fig 5
Lost messages of local commercial MQTT server

A commercial MQTT server is tested locally in evaluation version that allows a maximum of twenty-five simultaneous connections. One-hundred messages were transmitted successively by many clients simultaneously to the server, which is located on the same machine (a personal computer, Pentium Dual Core CPU at 3.2GHz and 2GB RAM), without inter-message delay. The results are shown in Table 6, and illustrated in Figure 5. With five clients, each transmits twenty messages successively; the server refused thirty-two messages. However, using an inter-message delay of ten milliseconds, zero-percent-losses were obtained. This explains the results of the previous experiments executed over the online-based servers, in tables 2, 3, 4, and 5. The network latency reacts as a delay between the successive messages, and hence, zero-percent-losses are obtained for the four online cases. Figure 6 illustrates the CPU usage while the commercial server was busy in replying to several clients, each sends one-thousand successive messages with ten milliseconds inter-message delay.

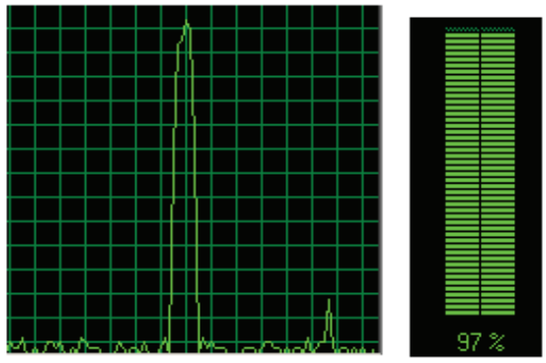

Fig 6

CPU usage while the server was busy in replying to thousands of MQTT messages.

Depending upon the previous comparison and these results, MQTT protocol is suitable for medium-to-big IoT business applications that use LANs and Intranets. However, the protocol is suitable for small-to-medium IoT business applications that use unreliable connections, mainly the Internet that suffers from varying periods of latency due to infrequent bandwidth limits. The MQTT's reliability is based on the connection-oriented telecommunication protocol, which is TCP, and the levels of services. Hence, Internet and wireless LANs (WLANs), which are unreliable connections, may employ MQTT in building reliable IoT applications. In the following sections, network model and topology were developed that employ the MQTT protocol in all levels of communicating entities, in order to build a small IoT application.

Network Model

In the network layer, and due to the varying bandwidth, unreliable wireless network, and limited resources of sensor systems, the communication protocol must be lightweight, easy to program, flexible, and reliable. Hence, the MQTT protocol can be used in different levels of communication infrastructure including sensor level, hub level, client level, and server level.

Concerning interoperability, MQTT uses UTF-8 encoding format (in binary representation). However, the protocol does not specify a way of object representations, and hence, interoperability is missed. JavaScript Object Notation (JSON) introduces good level of interoperability using text-based ASCII representation of objects (Bray, 2014). The JSON uses HTTP as a telecommunication protocol of the application layer. In this study, the JSON is adopted for the application layer of MQTT in order to enhance its interoperability.

The proposed network model is shown in Figure 7, where user data is formatted in JSON, transferred over MQTT using TCP port 1883, then over IP in the network layer, and over Wi-Fi or Ethernet in the physical layer. hence, in the application layer, the payload of the MQTT message, namely the PUBLISH packet, carries user data formatted in JSON. An example is illustrated in Figure 8, (a) shows the user data formatted in JSON, and (b) shows the payload of the PUBLISH packet formatted in UTF-8.

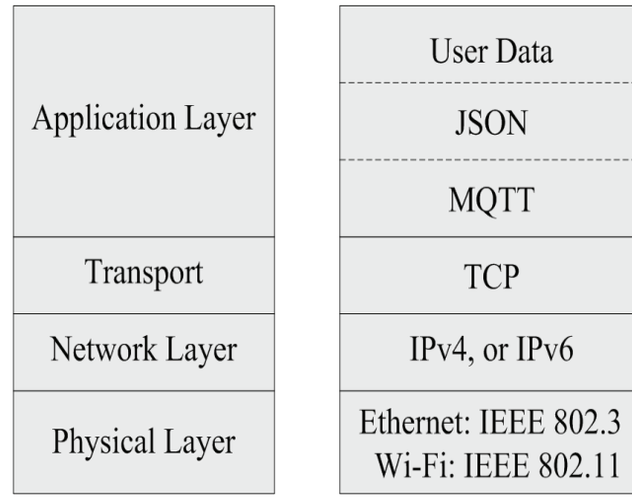

Fig 7

Proposed network model for the practical scenario

The model that appears in Figure 7 is used in the following section to develop a practical scenario based on star network-topology.

{"topic":"status/Temp","value":26.4}
(a)

7B 22 74 6F 70 69 63 22 3A 22 73 74 61 74 75 73 2F 54 65 6D 70 22 2C 22 76 61 6C 75 65 22 3A 32 36 2E 34 7D
(b)

Fig 8

(a) User data formatted in JSON, and (b) Payload of MQTT PUBLISH packet

Network Topology and Practical Scenario

Institutions and companies proposed some IoT hardware and software platforms, such as Arduino (arduino.cc), BeagelBone (beagleboard.org) and Raspberry (raspberrypi.org). Many IoT projects were developed based on Arduino platforms such as Uno, Mega, Nano, and Yun.

Arduino introduces both, the hardware and the integrated development environment (IDE) as well. BeagleBone Black (BBB) is another platform that supports IoT applications via its onboard Ethernet connection. The BBB, the Yun, and the Raspberry Pi platforms contain onboard microprocessor, and hence an embedded-Linux operating system. An embedded system, supported with an operating system empowers the platform with many advantages, including low cost, small size, portability, and low power consumption. Such systems are widely used for education and for small-businesses purposes.

A network topology is developed and its detailed diagram is shown in Figure 9. The network is in star mode for both the wired part and the wireless part, which uses Wi-Fi as a communication infrastructure. The "Raspberry PI 3" platform is proposed here as the MQTT server (broker), which is supported with a database management system, for storing and archiving the received messages from clients. The inexpensive Wi-Fi module, which is ESP8266, is used here as one of the IoT enabling and vital technologies. The module is microcontroller-based system, supported with TCP/IP full stack. Therefore, it enables direct communication between sensors and actuators, over the Internet or Intranet, with the MQTT servers (brokers). In addition, developers can use Arduino IDE to write software programs for the ESP8266 module. These characteristics make it a favorable platform for small applications with limited resources.

In spite of the high power consumption of Wi-Fi modules, the ESP8266 is used for IoT-based Smart City applications. Other Arduino platforms were employed, mainly the inexpensive Nano platform, which needs an add-on Ethernet module such as ENC28J60 and the comparatively expensive Arduino Yun platform, which contains built-in Ethernet and Wi-Fi, in addition to an embedded, Linux-based operating system. The Arduino Yun is a good choice for clustering MQTT servers, where more than one broker is needed for load balancing. Here, the Arduino Yun performs some MQTT brokering functions, with other MQTT-based clients. However, it reacts as an MQTT client as well for the main server, which is Raspberry PI 3.

MQTT clients, with user interfaces, such as the mobile phone, the tablet, and the laptop, are proposed here for monitoring sensors' measurements and control actuator-connected devices. The mobile-based client, subscribes to temperature (Temp) topic to get the latest sensor's readings, and controls the LED by publishing related commands. The opposite applies to the ESP8266, to the left, where the client publishes temperature status events, and to the right, where the client subscribes to LED control commands. In fact, MQTT excels also in mobile-based IoT applications, which requires simplicity and reliability.

Discussion

In this section, some topics such as scalability, security, and comparison with other publish-subscribe protocols, were discussed. Some of these topics are beyond the scope of this study; however, they represent important issues related to the functionality of the MQTT.

MQTT is broker based and may face performance and real-time response issues as system-scale increases, especially when the number of nodes and clients increases. In this case, MQTT servers, which are managed in clusters, and load balancers help solve such issues.

To build a trusted IoT-based environment, security issues must be considered. These include ways to protect connections, manage authentication, and ensure data confidentiality. In fact, the protocol provides simple username and password authentication, and SSL for data encryption. Refer to (HiveMQ, 2018) for more information on securing MQTT.

There are another three publish-and-subscribe protocols, namely advanced message queuing protocol (AMQP) (OASIS, 2012), simple text orientated messaging protocol (STOMP) (Github,

2012), and extensible messaging and presence protocol (XMPP) (Saint-Andre, 2011). Among these three protocols, XMPP is the only IETF standard. However, XMPP is an XML-based protocol developed for instant messaging, and not lightweight. AMQP is a protocol for the exchange of business transactions between two parties, which could complement MQTT in higher levels. STOMP is text-based, similar to HTTP, and not lightweight. Review (Piper, 2013), (Vasters, 2017), and (Luzuriaga et. al, 2015) for more information and comparisons.

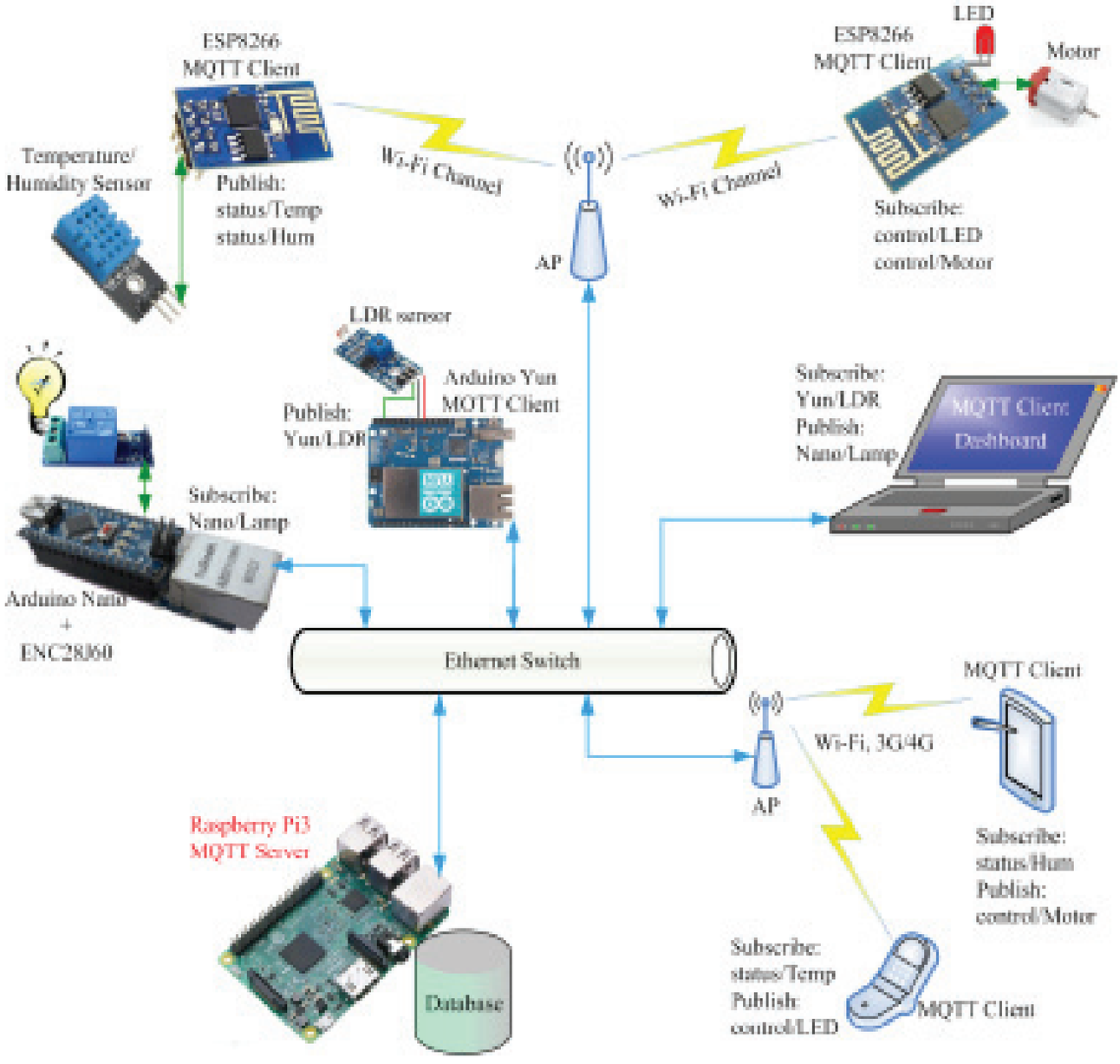

Fig 9

Network topology of a scenario that employs wired and wireless IoT platforms.

Conclusion

The telecommunication protocol, MQTT, is suitable for IoT applications in Smart Cities that have unreliable connections, mainly wireless networks, and internet. It is lightweight, easy to program, flexible, and reliable. In addition, it uses the publish-and-subscribe communication mechanism, where clients are loosely coupled, and are exchanging messages via a broker called, server. Depending upon the simulations conducted in this article, the MQTT is suitable for small-to-medium IoT-based business

applications of Smart Cities that use cloud-based servers (brokers) and for medium-to-big business applications that rely on LANs and Intranet. For example, Smart City applications such as energy monitoring, smart buildings, home automation and smart healthcare system may employ the MQTT protocol in all levels of communications. Hence, in this research article, the protocol is proposed for a complete practical scenario on different levels of communication infrastructures including sensor level, hub level, client level, and server level.

REFERENCES

1. Bray T. (Ed.) (2014, March). The JavaScript Object Notation (JSON) Data Interchange Format. Internet Engineering Task Force, Request for Comment 7159. Retrieved December, 03, 2017, from http://tools.ietf.org/html/rfc7159

2. Fielding R. and Reschke J. (Eds.) (2014, June). The Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. Internet Engineering Task Force, Request for Comment 7230. Retrieved December, 03, 2017, from http://tools.ietf.org/html/rfc7230

3. Github, (2012, October). STOMP Specifications version 1.2. Github. Retrieved March, 03, 2018, from https://stomp.github.io/

4. HiveMQ, (2018, January). MQTT Security Fundamentals. HiveMQ. Retrieved March, 03, 2018, from https://www.hivemq.com/mqtt-security-fundamentals/

5. ISO (2016, July). MQTT 3.1.1 Specifications. International Organization for Standardization, ISO/IEC 20922. Retrieved December, 03, 2017, from https://www.iso.org/standard/69466.html

6. Jaloudi, Open source software of Smart City protocols current status and challenges, international conference on open source software compntiny (OSSCOM), 2015, Amman, Jordan, pp. 1-6, 2015.

7. Joshi J. et. al, Performance enhancement and IoT based monitoring for smart home, Information Networking (ICOIN), 2017 IEEE International Conference on, Da Nang, Vietnam, pp. 468-473, 2017

8. Kang D. H. et. al, Room Temperature Control and Fire Alarm/Suppression IoT Service Using MQTT on AWS, Platform Technology and Service (PlatCon), 2017 IEEE International Conference on, Busan, South Korea, pp. 1-5, 2017

9. Khoi N. M. et. al, IReHMo: An efficient IoT-based remote health monitoring system for smart regions, E-health Networking, Application & Services (HealthCom), 2015 17th IEEE International Conference on, Boston, MA, USA, pp. 563-568, 2015

10. Kodali R. K. and Soratkal S., MQTT based home automation system using ESP8266, Humanitarian Technology Conference (R10-HTC), IEEE Region 10, Agra, India, pp. 1-5, 2016

11. Locke D., (2010, July). MQ Telemetry Transport (MQTT) V3.1 Protocol Specification. IBM. Retrieved December, 03, 2017, from https://www.ibm.com/developerworks/library/ws-mqtt/

12. Lohokare et. al, Scalable tracking system for public buses using IoT technologies, Emerging Trends & Innovation in ICT (ICEI), 2017 IEEE International Conference on, Pune, India, pp. 104-109, 2017

13. Luzuriaga et. al, A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks, Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE, Las Vegas, NV, USA, pp. 931-936, Jan., 2015

14. OASIS (2014, October). MQTT version 3.1.1 Specifications. Organization for the Advancement of Structured Information Standards. Retrieved December, 03, 2017, from http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

15. OASIS (2012, October). AMQP version 1.0 Specifications. Organization for the Advancement of Structured Information Standards. Retrieved March, 11, 2018, from http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

16. Oryema B. et. al, Design and implementation of an interoperable messaging system for IoT healthcare services, Consumer Communications & Networking Conference (CCNC), 2017 14th IEEE Annual, Las Vegas, NV, USA, pp. 45-52, 2017

17. Piper A., (2013, Feb.). Choosing Your Messaging Protocol: AMQP, MQTT, or STOMP. VMware. Retrieved March, 11, 2018, from https://blogs.vmware.com/vfabric/2013/02/choosing-your-messaging-protocol-amqp-mqtt-or-stomp.html

18. Saint-Andre P., (2011, March). Extensible Messaging and Presence Protocol (XMPP): Core. Internet Engineering Task Force, Request for Comment 6120. Retrieved March, 11, 2018, from https://tools.ietf.org/html/rfc6120

19. Sharad et. al, The smart bus for a smart city — A real-time implementation, Advanced Networks and Telecommunications Systems (ANTS), 2016 IEEE International Conference on, Bangalore, India, pp. 1-6, 2016

20. Shelby Z., Hartke K., and Bormann C. (2014, June). The Constrained Application Protocol (CoAP). Internet Engineering Task Force, Request for Comment 7252. Retrieved December, 03, 2017, from http://tools.ietf.org/html/rfc7252

21. Thota P. and Kim Y., Implementation and Comparison of M2M Protocols for Internet of Things, Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (IEEE ACIT-CSII-BCD), Las Vegas, NV, USA, pp. 43-48, 2016

22. Yi D. et. al, Design and implementation of mobile health monitoring system based on MQTT protocol, Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), 2016 IEEE, Xi'an, China, pp. 1679 - 1682, 2016

23. Yokotani T. and Sasaki Y., Comparison with HTTP and MQTT on required network resources for IoT, Control, Electronics, Renewable Energy and Communications (ICCEREC), 2016 IEEE International Conference on, Bandung, Indonesia, pp. 1-6, 2016

24. Vasters C., (2017, Jan.). From MQTT to AMQP and back. A model for integration of the two most popular open messaging protocols leveraging the greatest strengths of both. Vasters. Retrieved March, 11, 2018, from http://vasters.com/blog/From-MQTT-to-AMQP-and-back/