ISSN XXXX XXXX © 2018 IJESC

Research Article Volume 8 Issue No.12

Evaluation of MQTT Protocol for IoT Based Industrial Automation

Hamid M. Hasan[1], Bahaa K. mohammed[2]
Control and Systems Engineering Department University of Technology, Baghdad, Iraq

Abstract:
Currently, the significant changes in the control of industrial processes, intelligent building control and automation techniques are under pressure to reduce operating costs and integrate the important advances of communications software. IoT has become a key factor in enterprise-wide production and enterprise systems. The Internet connection has a fundamental change in the control arrangements, and the use of open / public and personal standards that make computer systems (computers, tablets and smart phones) achieve great benefits for its users and producers. This led to the study of the protocols of the internet of things IoT. In this work, we want to evaluate the most famous application level protocol MQTT in terms of using it in industrial environments.
To achieve the full evaluation of MQTT protocol, a suitable simulation platform was required. Through extensive research, the OMnet++ platform with INET framework was selected. Using this platform, simulation can cover a wide range of network layer protocols, topologies, architectures, and communication parameters. It's observed that the throughput which obtained for the subscriber with respect to the number of sensors, is increased steadily at the first 40 sensors. Then it goes down due to the wide range of parameters that distributed over the whole network.

Keywords: INET, IoT, MQTT, OMnet++, WSN.

1. IOT BASED INDUSTRIAL AUTOMATION

As a result of the development and low cost of microcontrollers and telecommunications on the one hand, and the orientation of industrial enterprises (the development of industry on an extensive scale) and large companies to a high-level, modern data-gathering approaches and the compilation of information from sprawling workplaces for the purpose of analysis and appropriate decision-making, on the other hand, the demand for the Internet of Things has become increasingly important in recent years.
The Internet of Things (IoT) is the best way to connect industrial devices and sensors, some of them via the Internet, allowing an authorized user in the industry to use the information from these connected devices to process data obtained in a useful way. IoT applications usually support, collect, analyze, and visualize data. The internet of things structure includes the latest technologies such as computers, smart devices, telecommunications, and cloud computing. Bluetooth and RF (radio frequency) techniques were used in the past to control industrial applications but were limited to a short distance. The operator must be in a Bluetooth connection range or in the radio frequency Zone. The solution for short-range communication is to automate the IoT-based industry stuff. Here one can control and survey from anywhere in the world. [1]

1.1 PLATFORM OPERATING SYSTEMS

Several types of platform operating systems are used to perform IoT functions and several types of real-time operating systems (RTOS), which are factors for the growth of RTOS-based IoT. Contiki RTOS is an example of the widely used operating systems in IoT. Contiki supports many protocols including MQTT (the subject of this thesis) and is applicable to the platforms mentioned above [2]. Other operating systems such as TinyOS [3], LiteOS [4] and Riot OS [5] are lightweight and dedicated to IoT platforms. Some automakers have also been influencing Google to set up the Open Auto Alliance (OAA) to develop an Android platform architecture to keep up with the IoV model [6].A comparison of some structures of these operating systems is listed in Table I. [7]

TABLE I
COMMON OPERATING SYSTEMS USED IN IOT ENVIRONMENTS

Operating System	Language Support	Minimum memory (kB)	Event-based	Operating System	Language Support
Tiny OS	nesC	1	Yes	Partial	Yes
Contiki	C	2	Yes	Yes	Yes
LiteOS	C	4	Yes	Yes	Yes
Riot OS	*C/C++*	1.5	No	Yes	Yes
Android	Java	-	Yes	Yes	Yes

2. MQTT PROTOCOL

MQTT is an abbreviation of (Message Queuing Telemetry Transport) which is an application level protocol that operates on top of the TCP / IP stack. It is easy, lightweight and easy to enforce protocol which utilizes client-server using publish-subscribe messaging sample. The reliable and fashionable states of the protocol, make it the best to apply in contexts like M2M (Machine to Machine) or IoT in which a small code footprint is required [8].

2.1 PUBLISH-SUBSCRIBE PATTERN

The pattern of the system is publish-subscribe which relies on a central node called broker that acts as a Server, the Clients connect to it and exchange messages through. In this pattern, clients are not aware of each other as they see the connection with the broker only. The functionality of the broker is to deliver the messages of the sender clients to the recipient ones. The logic of messages delivering is based on topics, sender clients assign their messages with a specific topic. Brokers do maps of the recipient with its interest one or more topics. With this information, Broker will be able to deliver senders messages according to the attached topic, to its registered recipients on that topic [9].

2.2 TOPICS

The topics in MQTT protocol are formed as hierarchical level, using (/) as a split code similar to that used in HTTP protocol in URLs request message. There are two types of wildcards used in MQTT, single and multilevel, the first is denoted by (+) sign, and the other uses (#) sign. The following is an example of using the single level:

level1/ + /level3 equivalent with level1/ X /level3 and level1/ Y /level3.

In case of multilevel:

Level1/level2/ # equivalent with level1/level2/ X/ Y and level1/level2/ Y/ X [10].

2.3 QUALITY OF SERVICE

Quality of service in MQTT protocol is the message delivering assurance between clients and broker, there are three QoS levels: (0, at most one), (1, at least one) and (2, exactly once) [10].

- QoS0

Leve0 QoS is the simplest one, in which the publish packet, whether from the publisher to the broker or from the broker to the subscriber, wait for no acknowledgment. So, in QoS 0 PUBLISH packet will be received at most once [10].

- QoS1

In level1 QoS, PUBLISH packet will be received at least once, in which, every PUBLISH packet waiting to be acknowledged by the recipient with PUBACK packet. If no confirmation has been received in a specific window of time then the PUBLISH packet will be sent again, and this may happen several times. [10]

- QoS2

In QoS of level2, PUBLISH packet is ensured by the receiver exactly once. There are at minimum four packets which will be exchanged between both the sender and the recipient. The first one is the PUBLISH packet. To prevent processing the same PUBLISH packet by the receiver twice, it will store a reference to this packet by using a packet identifier. PUBREC packet containing original packet identifier will be sent as acknowledgment upon the PUBLISH packet. When the sender receives a PUBREC packet indicating that the receiver had successfully received it, it will discard the original PUBLISH packet. By receiving the PUBREC acknowledgment, the sender store a reference and sends PUBREL that contains the original packet identifier. When the receiver receives the PUBREL packet, it will discard all the states that are related to the identifier and send PUBCOMP notifying the sender that the message had been completely received [10].

2.4 SECURITY

During connection establishment, CONNECT packet carries the username, password and client identifier. By using credentials, the broker will authenticate the client and authorize the related topics and other resources. The broker should be configured to the proper setting before allowing the client to be connected. This information was sent as plaintext within the CONNECT packets, therefore it will be visible to the intermediate network nodes when transfer operation is done on plaintext TCP. For this reason, the CONNECT packet should be sent on a protocol that provides confident data transfer. TSL is one of these underlying protocols, where it has an algorithm of encryption and integrity checking, that can be applied to MQTT packets. Furthermore, each of the broker and the client can authenticate itself to the other using X.509 certificates. X.509 in contrast to plaintext, provides a better authentication method for the involved parties [10].

2.5 MQTT CONTROL PACKETS

There are fourteen control packet types defined in MQTT standard. Table II lists these packet types with a brief description.

Each one of MQTT packets can be divided into three parts. The parts are fixed header, variable header, and payload. The part that presents in all packets is the fixed header, while the other two parts will vary as needed. It is organized as mentioned above, the fixed header first, the variable header is the second then payload [6]. The fixed header consists of a set of fixed fields, while the other two parts have a varying fields, according to the type that it was built for.

There are three fields in fixed header; the packet type, the flags and remaining length. The first four most significant bits belong to the package type and the flags reserve the other four bits from the first byte. Packet type is represented by a numerical form. The second field represents specific flags for each package (not used in most cases). As for the remaining length, it represents the number of bytes in the rest of the packet, except for the fixed header [6].

TABLE II
MQTT STANDARD CONTROL PACKETS

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Serverto Client	Publish acknowledgment
PUBREC	5	Client to Server or Serverto Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Serverto Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Serverto Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Serverto Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Serverto Client	Unsubscribeackno wledgment

PINGREQ	12	Client to Server	PING request
PINGRESP	13	Serverto Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

3. Tools And Methodology

In a traditional industrial IoT architecture there are one or more sensors, actuators and controllers. These typically connect to an edge gateway and the edge gateway talks across the wide area network (typically the internet) to the cloud or enterprise or server. There are a number of variations, for instance where a device is formed from a combination of sensors and a gateway. Figure (3-1) show the traditional industrial IoT network architecture

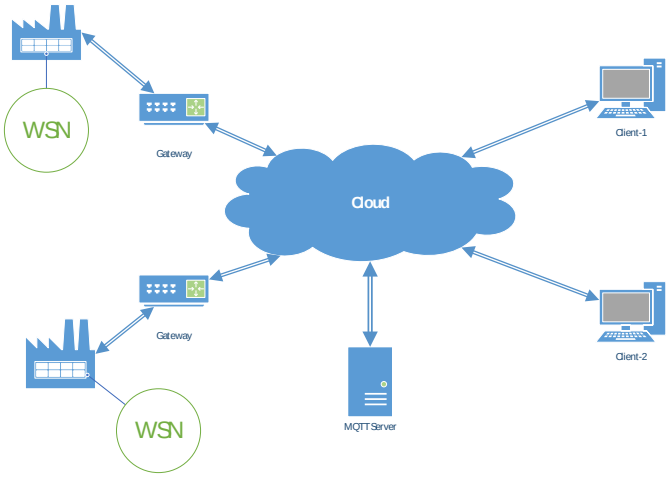


Figure (3-1). Industrial IoT network architecture

3.1 Evaluation Method

For the purpose of evaluating the performance of the MQTT protocol, several network scenarios have been designed in the base of the industrial environment. In order to carry out this work, it was necessary to create the needed modules, in addition to the modules that already exist in the INET framework. As the working environment is OMNet++, as shown in Figure (3-2), the procedure should be followed in accordance with the methodology of the work in this environment.

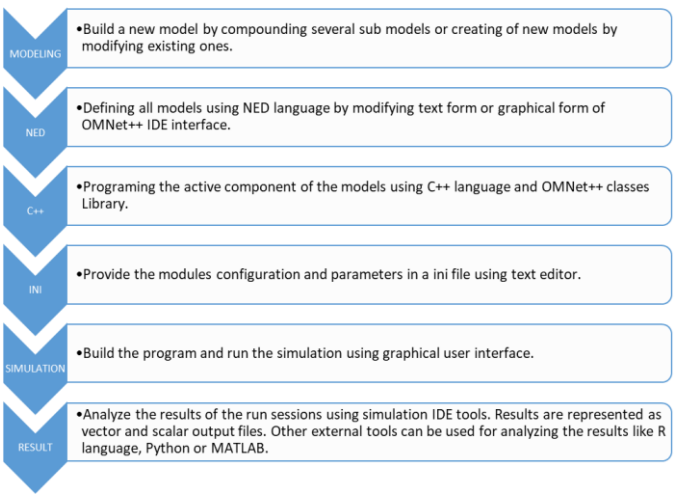


Figure (3-2). OMNet++ using procedure.

3.2 The Network Module

The basic network is implemented to contain the major branches connected with the internet cloud module. These branches are the sender, recipient, and server, as shown in Figure (3-3).a. Sender represents industrial plant side which consists of sensors, actuators, and gateway, as shown in Figure (3-3).b. Recipient represents the control and monitoring sides which consist of the hosts acting as subscribers. The third branch is the server which represents the broker in the case of MQTT protocol.

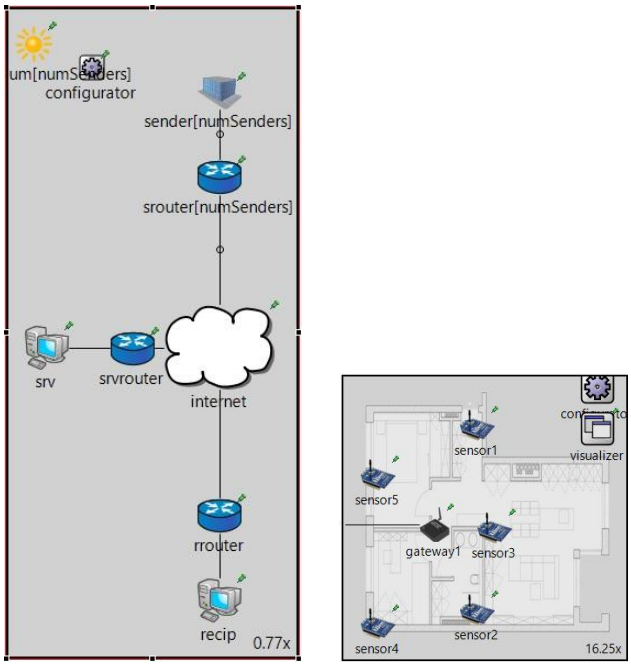


Figure (3-3). a. Sender represents industrial plant side.
b. Recipient represents the control and monitoring sides.

3.3 Broker Module

The code of broker is focused on three basic functions for the purpose of simulation and evaluation that are related with the subject of this thesis. The broker functions are summarized in the following points:

- Perform authenticated connection for both Publisher and Subscriber.
- Accept Subscriber registrations and map them to its interested topics.
- Receive published data and forward it to its topic Subscribers.

3.4 Publisher Module

All applications implement the IApp module interface to ease configuring the StandardHost module. MQTT is based on TCP protocol, so the publisher application is programed as a TCP application module. TcpSocket.cc module is used as the interface between the application module and TCP.

3.5 Subscriber Module

Similer to the Publisher, the Subscriber is a TCP client application module. Its behavior is, when TCP connection established, the subscription message will be sent to the Broker. The Broker in turn sends an acknowledgment to accept the registration. After that the Subscriber will be ready to receive the data message published from Broker.

3.6 MODULES PARAMETERS

The network node modules used to implement the simulation are listed below in Table III.

Table III.
Simulation Network nodes.

Node name	Description
Sender	Plant area
Sensor	Publisher node
Actuator(lamp)	Subscriber node
Srv	Broker
Recip	Subscriber node
Recip1	Publisher node
Internet	Internet cloud
Router	Router node

Parameters of the sensor node module are configured in (.ini) file, as shown in Figure (3-4). Where (TCPMqttclientApp) is the publisher module that is assigned to the sensor node, (dataLength) represents the packet length in bytes and (senddelay) is the waiting interval of sensor for each send operation.

```
**.sensor*.numApps = 1
**.sensor*.app[*].typename = "TCPMqttClientApp"
*.sender[*].sensor*.app[0].localAddress = ""
*.sender[*].sensor*.app[0].localPort = -1
*.sender[*].sensor*.app[0].connectAddress = "srv"
*.sender[*].sensor*.app[0].connectPort = 80
*.sender[*].sensor*.app[0].startTime = exponential(5s)
*.sender[*].sensor*.app[0].numRequestsPerSession = 100
*.sender[*].sensor*.app[0].requestLength = 12B
*.sender[*].sensor*.app[0].dataLength = 12

*.sender[*].sensor*.app[0].replyLength = exponential(2000B)
*.sender[*].sensor*.app[0].thinkTime = truncnormal(5s,10s)
*.sender[*].sensor*.app[0].idleInterval = 0s
*.sender[*].sensor1.app[0].senddelay = 1s
*.sender[*].sensor2.app[0].senddelay = 5s
*.sender[*].sensor3.app[0].senddelay = 10s
*.sender[*].sensor4.app[0].senddelay = 20s
*.sender[*].sensor5.app[0].senddelay = 40s
```

Figure (3-4). Sensor node parameters.

Actuator nodes are configured as shown in Figure (3-5). Where (TCPMqtt-SubClient) is the subscriber module and topicid is the topic that is assigned to each actuator.

```
*.sender[*].lamp*.numApps = 1
*.sender[*].lamp*.app[*].typename = "TCPMqttSubClientApp"
*.sender[*].lamp*.app[0].localAddress = ""
*.sender[*].lamp*.app[0].localPort = -1
*.sender[*].lamp*.app[0].connectAddress = "srv"
*.sender[*].lamp*.app[0].connectPort = 80
*.sender[*].lamp*.app[0].startTime = exponential(5s)
*.sender[*].lamp*.app[0].numRequestsPerSession = 1
*.sender[*].lamp*.app[0].requestLength = 12B
*.sender[*].lamp*.app[0].replyLength = exponential(10B)
*.sender[*].lamp*.app[0].thinkTime = truncnormal(5s,10s)
*.sender[*].lamp*.app[0].idleInterval = 0s
*.sender[*].lamp1.app[0].topicid=300
*.sender[*].lamp2.app[0].topicid=301
*.sender[*].lamp3.app[0].topicid=302
*.sender[*].lamp4.app[0].topicid=303
*.sender[*].lamp5.app[0].topicid=304
*.sender[*].lamp6.app[0].topicid=305
*.sender[*].lamp7.app[0].topicid=306
*.sender[*].lamp8.app[0].topicid=307
```

Figure (3-5). Actuator node paramiters.

The parameter of the (recip) module is configured as shown in Figure (3-6).

```
**.recip.numApps = 1
**.recip.app[*].typename = "TCPMqttSubClientApp"
**.recip.app[0].localAddress = ""
**.recip.app[0].localPort = -1
**.recip.app[0].connectAddress = "srv"
**.recip.app[0].connectPort = 80
**.recip.app[0].startTime = exponential(5s)
**.recip.app[0].numRequestsPerSession = 1
**.recip.app[0].requestLength = 12B
**.recip.app[0].replyLength = exponential(10B)
**.recip.app[0].thinkTime = truncnormal(5s,10s)
**.recip.app[0].idleInterval = 0s
**.recip.app[7].topicid = 200
```

Figure (3-6). (recip) module parameters.

Lastly, the (recip1) module parameters are configured in the ini file as shown in Figure (3-7). (recip) is a publisher module, it must have number of apps equals to the actuator nodes.

```
**.recip1.numApps = 8
**.recip1.app[*].typename = "TCPMqttClientApp"
**.recip1.app[*].localAddress = ""
**.recip1.app[*].localPort = -1
**.recip1.app[*].connectAddress = "srv"
**.recip1.app[*].connectPort = 80
**.recip1.app[*].startTime = exponential(5s)
**.recip1.app[*].numRequestsPerSession = 1
**.recip1.app[*].requestLength = 12B
**.recip1.app[*].replyLength = exponential(10B)
**.recip1.app[*].thinkTime = truncnormal(5s,10s)
**.recip1.app[*].idleInterval = 0s
**.recip1.app[*].dataLength = 12
**.recip1.app[0].senddelay = 10s
**.recip1.app[0].topicid = 300
**.recip1.app[1].senddelay = 10s
**.recip1.app[1].topicid = 301
**.recip1.app[2].senddelay = 20s
**.recip1.app[2].topicid = 302
**.recip1.app[3].senddelay = 30s
**.recip1.app[3].topicid = 303
**.recip1.app[4].senddelay = 40s
**.recip1.app[4].topicid = 304
**.recip1.app[5].senddelay = 50s
**.recip1.app[5].topicid = 305
```

Figure (3-7). (recip1) node parameters.

3.7 IMPLEMENTATION

The new modules that created are placed in the application layer directory of the Inet4 framework as shown in Figure (3-8). MQTT protocol main directory is named (mqtt) where the broker and clients (Publisher and Subscriber) modules are placed in the sub directories (broker) and (client) respectively. Common Directory contains the message and serializer modules.

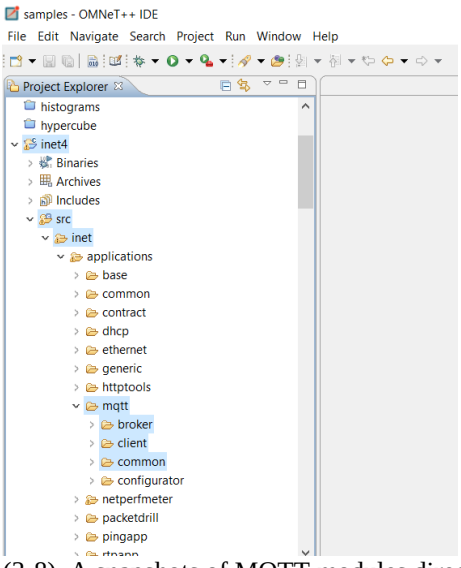
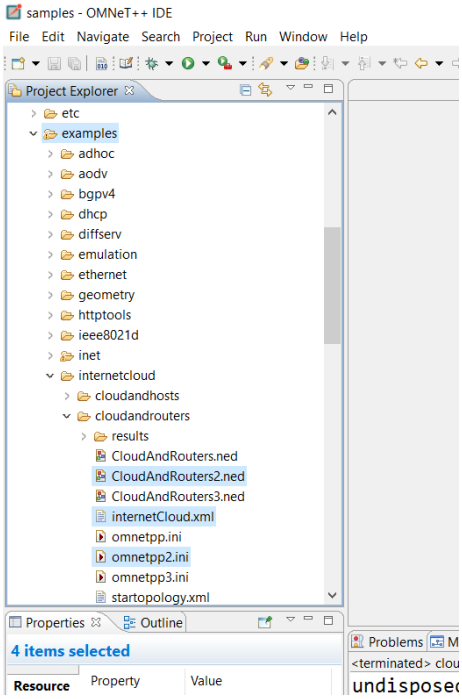


Figure (3-8). A snapshots of MQTT modules directories.

The test-implementation and results gathering files are an examples that placed in the examples directory of Inet4 framework. This example was created by modifying an existing example as shown in Figure (3-9). The related files

are (CloudAndRouters2.ned, internetCloud.xml and omnetpp2.ini).



Figure(3-9). A snapshots of example files.

For a specific time limit duration the (RunUntil) icon can used. In this case simulation can be stoped at a time or event number that optionally selected by the user. Running speed can also be set to the specific level using Running mode option as shown in Figure(3-10).
A several snapshots have been token at the run time as shown in the Figures(3-11,12,13)

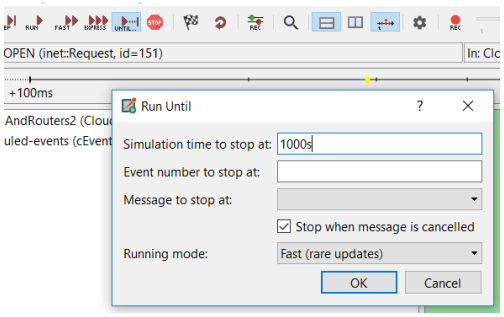


Figure (3-10). A snapshot of Run Until window.

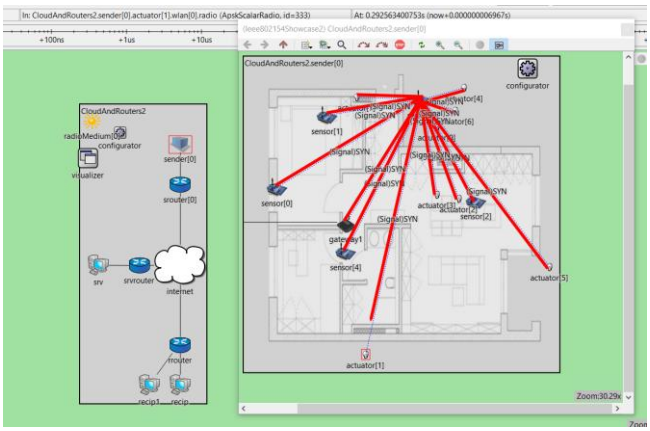


Figure (3-11). A snapshot of WSN activity.

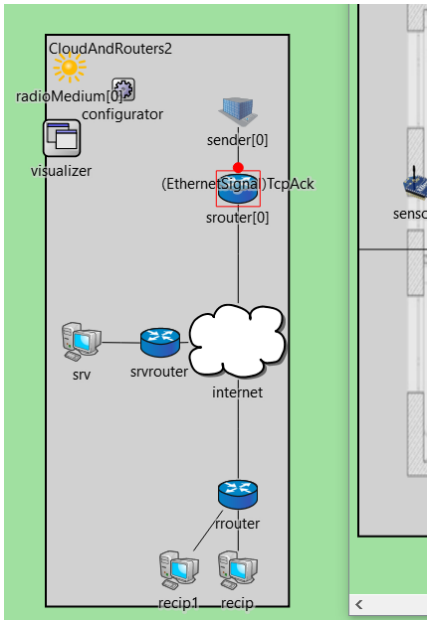


Figure (3-12). A snapshot of sneder activity.

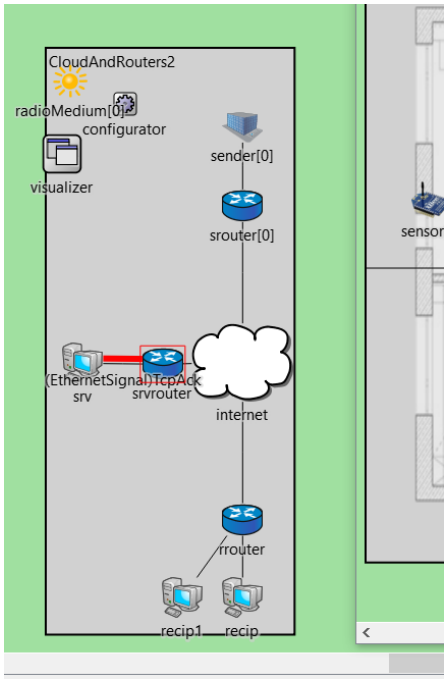


Figure (3-13). A snapshot of broker activity.

4. RESULTS

Network defined file was modified to change the number of sensor nodes to obtain its impact on the network evaluation methods. Several runs are required, thus the NED file must be modified to make dynamic change of sensors and actuators, as shown in Figure (4-1).

Figure (4-1). Sensors and actuators input number.

4.1 THROUGHPUT

The change of the throughput with respect to the number of the nodes has been obtained for the subscriber as shown in Figure (4-2).

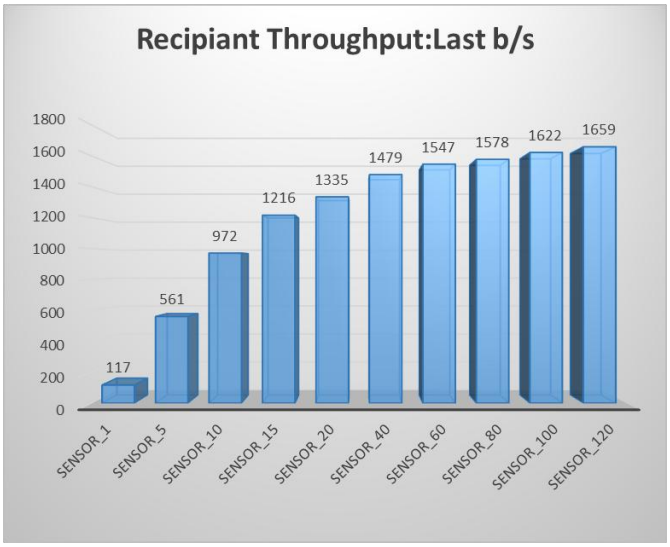


Figure (4-2). Subscriber throughput

4.2 END TO END DELAY

The impact of the sensor number increasing in the case of end to end delay at the subscriber is shown in Figure (4-3).

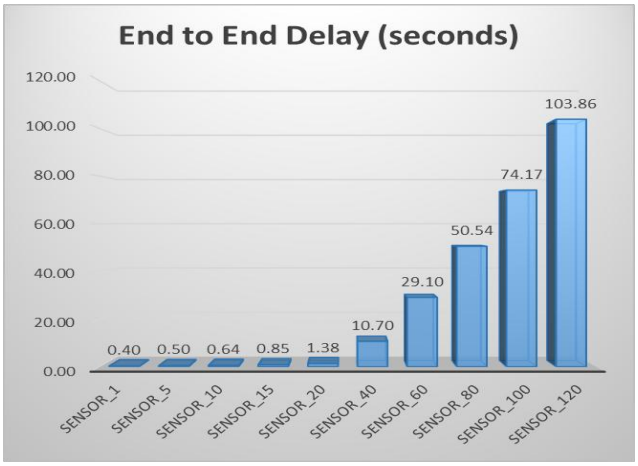


Figure (4-3). End to end delay.

4.3 GOODPUT

Goodput has been obtained by dividing of subscriber's packet count over the summation of the packets sent by all sensor nodes. Figure (4-4) shows the impact of the sensor node increasing on the goodput value.

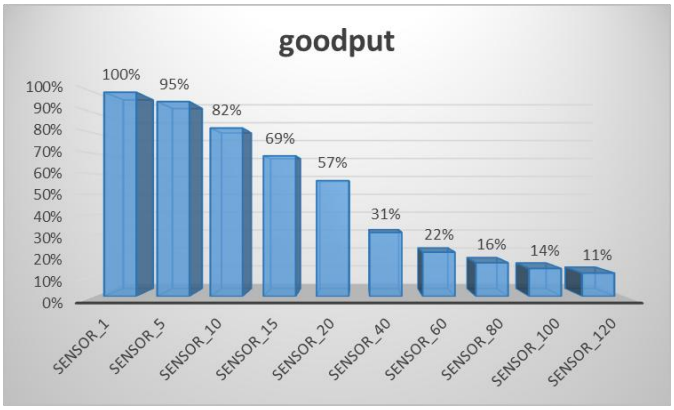


Figure (4-4). Goodput change with sensor nodes increasing.

5. CONCLUSION

It's observed that the throughput which obtained for the subscriber with respect to the number of sensors, is increased steadily at the first 40 sensors. Then it goes down due to the wide range of parameters that distributed over the whole network. Some of these parameters are internet cloud packet drop ratio, routers queuing, radio medium bitrate error ratio and WSN protocol. Also it's observed how the end to end delay will increases with the number of sensors.

REFERENCES

[1] P. N. M, N. Madhukar, A. Ashwini, J. Muddsar, and M. Saish, "IOT Based Industrial Automation," no. Acbcda 2017, pp. 36–40.

[2] A. Dunkels, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors."

[3] P. Levis *et al.*, "TinyOS: An Operating System for Wireless Sensor Networks," *Ambient Intell.*, vol. 8491, pp. 115–148, 2005.

[4] Q. Cao, T. Abdelzaher, J. Stankovic, and T. He, "The LiteOS Operating System: Towards Unix Like Abstraction for Wireless Sensor Networks," *Proc. 7th Int. Conf. Inf. Process. Sens. Networks (IPSN 2008)*, 2008.

[5] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," *2013 IEEE Conf. Comput. Commun. Work. (INFOCOM WKSHPS)*, pp. 79–80, 2013.

[6] Openautoalliance.net, "OPEN AUTOMOTIVE ALLIANCE," 2015. [Online]. Available: https://www.openautoalliance.net/#about.

[7] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[8] A. Banks and R. Gupta, "MQTT Version 3.1.1 Plus Errata 01," no. December, 2015.

[9] P. T. H. Eugster, P. a Felber, and A. Kermarrec, "The Many Faces of Publish / Subscribe," *Computing*, vol. 35, no. 2, pp. 114–131, 2003.

[10] D. Locke, "Introduction to MQTT," *Webinar Presenters*, 2013. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt.