

Performance Analysis of QoS for the MQTT-SN Protocol with Industry Oriented MQTT-SN Gateway and Integration with Cloud MQTT-Server, IOT-Application

M.ObulaReddy¹, Dr.J.B.Seventline²

¹Research Scholar, Department of Electronics and Communication Engineering, Gitam Deemed to be University, Vishakapatnam, India, moredy2003@gmail.com

²Professor, Department of Electronics and Communication Engineering, Gitam Deemed to be University, Vishakapatnam, India seventline.joseph@gitam.edu

Abstract

Internet is a global communication network to provide various services like file transfer, email and other services using various heterogeneous application messaging protocols HTTP, MQTT, COAP, DDS, AMQP. Each of these application messaging protocols are designed and implemented as per different application requirements considering the computational resources and available communication bandwidth. These protocols are not suitable for constrained sensor devices due to limitation of computational power and bandwidth. MQTT protocol is designed and implemented for Machine to Machine communication, but still not suitable for low power sensor devices. More Efficient MQTT-SN protocol is (Message Queue Telemetry Transport-Sensor Network) proposed for sensor devices considering the wireless sensor network characteristics, power constraint and bandwidth limitations. In this paper we discussed MQTT-SN protocol important features, MQTT-SN QoS impact analysis for the IOT applications, End to End delay (Sensor Node to IOT Application) calculations, message overhead analysis for MQTT-SN, MQTT, COAP protocols and MQTT-SN Gateway Integration with Industry oriented Cloud MQTT-Server.

Keywords— IOT, MQTT-SN, TCP, UDP, MQTT, HTTP, COAP, WIRESHARK

I. INTRODUCTION

Current wireless sensor networks are designed and developed for various applications like home automation, smart cities, environmental monitoring, structural health monitoring etc using either preparatory application messaging protocols or incompatible protocols with current wide spread internet communication protocols. Currently various application messaging protocols are designed and developed for internet communication like MQTT, COAP, XMPP, DDS and HTTP. These protocols are not suitable for constrained sensor devices due to low power and bandwidth limitation. More Efficient Application protocol needed for sensor devices considering lossy wireless network, low power and bandwidth limitation. Message Queue Telemetry Transport -Sensor Network (MQTT-SN) protocol is right choice protocol for Sensor Devices due to Low message overhead compared other available messaging protocols. MQTT-Message Queue Telemetry Transport protocol designed and developed for machine to machine communication. MQTT is light weight protocol, but underlying transportation mechanism used as TCP/IP. TCP transport protocol too complex for low power sensor devices. MQTT-SN uses UDP/IP Transport communication protocol. UDP is light weight compared with TCP/IP. As Per [1] MQTT-SN full detailed design specification is mentioned. Some of the MQTT-SN important features are discussed in Section III. As per [2], Theoretical comparison of IOT messaging protocols are discussed in terms message overhead, throughput and bandwidth. As per [3] MQTT-SN End to End delay performance simulated using NS-2 Simulator. The organization of the paper as follows: In Section II, IOT Architecture [Sensor Node, IOT Gateway, MQTT Server, and IOT Application] is explained. Section III discusses important MQTT-SN protocol features, MQTT-SN messages description, MQTT-SN QoS model and MQTT-SN Topic management, message flows for different QoS. In Section IV discusses IOT application development process. In Section V discusses Experimental setup hardware and Message overhead, End to end Delay analysis, message loss analysis. In Section VI, Important trace logs are

discussed for all nodes (Sensor Device, IOT Gateway, MQTT Server, and IOT Application). In Section VII results and conclusions are explained.

II. IOT SYSTEM ARCHITECTURE

Overall IOT System architecture shown in Fig -01

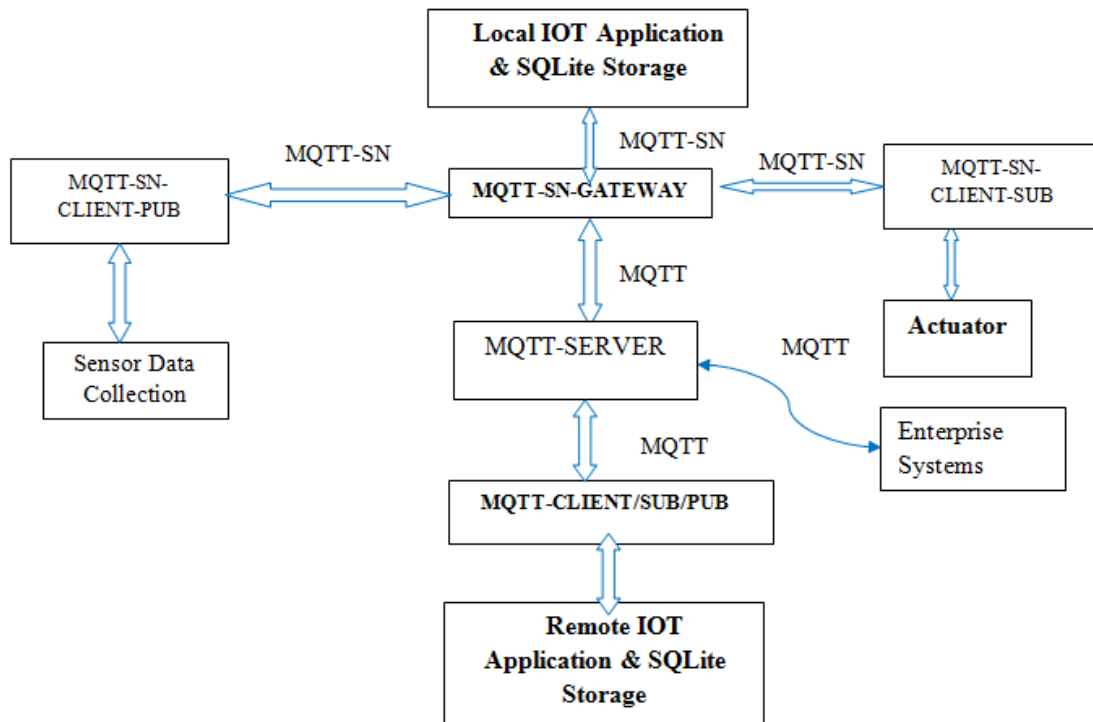


Fig 01: IOT System architecture

A. Sensor Node Platform

Sensor devices capture the sensor data with predefined time interval. Sensor Node platform pack the sensor data with the JSON data format and Publish Packed sensor data to the IOT MQTT-SN Gateway with pre-configured time interval. Sensor Node platform as shown in Fig-02

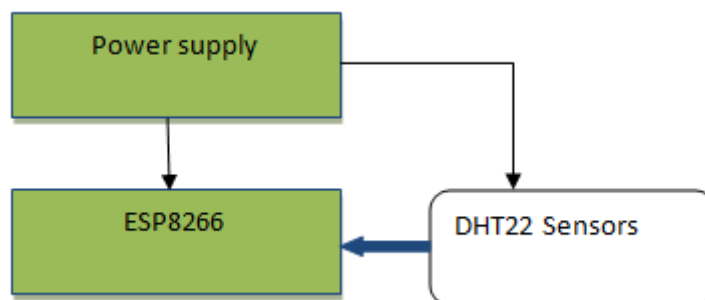


Fig 02: Sensor Node IOT platform

B. MQTT-SN Gateway

MQTT-SN Gateway is crucial computing Node in the IOT Architecture. Main function of MQTT-SN Gateway is receives the MQTT-SN messages from sensor devices and Translate to the MQTT Messages as per MQTT Protocol specifications. MQTT-SN (Message Queue Telemetry

Transport-Sensor Network) and MQTT protocols are different protocols, but it is closely related. MQTT-Gateway designed and developed with four multi-threaded tasks-MQTT-SN Receiver Task, MQTT-SN Sender Task, MQTT Sender Task, MQTT Receiver Task and Event Queue Manager.

MQTT-SN Receiver Task: It receives MQTT-SN messages from sensor devices, decode the messages and store the message parameters in the Event Queue Manager for sending to the MQTT-Server and also maintain MQTT-SN Receiver Process State machine.

MQTT Sender Task: It takes the MQTT-SN parameters from the Event Queue Manager and Encode to the MQTT Messages, send to the MQTT-Server as per MQTT Protocol Specification and maintain the **MQTT Sender Process State machine.**

MQTT Receiver Task: It receives MQTT messages, decode the MQTT Messages and store the relevant message parameters in the event queue manager.

MQTT-SN Sender Task: It takes the MQTT parameters from the event queue manager and encode to the MQTT-SN Messages, send to the sensor IOT Platform as per MQTT-SN Protocol Specification and maintain the **MQTT-SN Sender process state machine.**

Gateway protocol software process diagram as shown in Fig -03

C. MQTT Server

MQTT- Message Queue Telemetry Transport protocol is light weight protocol designed and developed for machine to machine communication devices. MQTT Server receives the publish messages from the MQTT-SN Gateway and publishes the messages to the Subscribed devices. Publish and subscribe Mechanism is an asynchronous process. In this project MQTT-MOSQUITTO server used for Gateway integration.

D. IOT Application

IOT Application is any type of application for taking appropriate action, control the actuator devices and monitoring sensor/actuator devices. Sensor devices data stored in the SQL lite data base for the future processing.

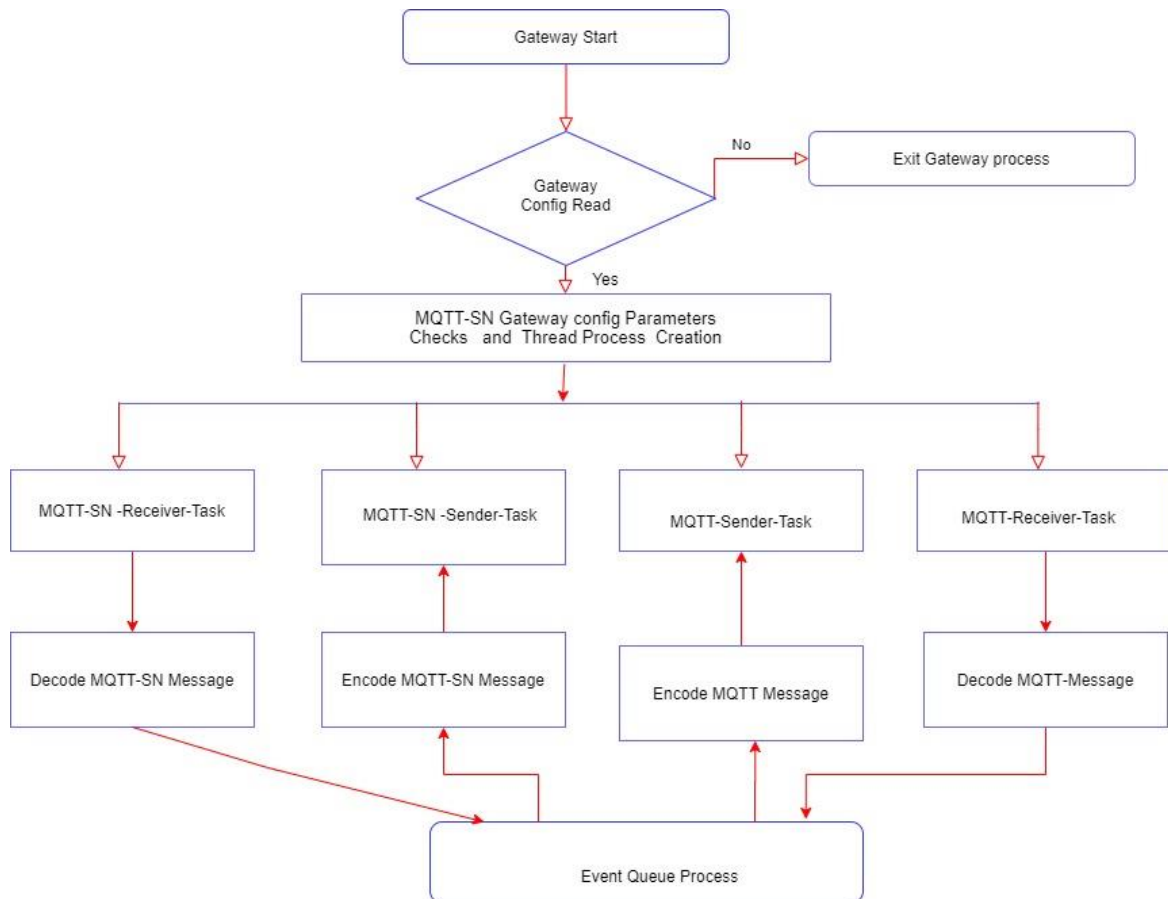


Fig 03: Gateway Software Architecture

III. MQTT-SN NEW FEATURE COMPARED WITH MQTT PROTOCOL

A. Gateway Advertise feature

GW Advertise service periodically broadcast the gateway info to the clients. Sensor devices decode the gateway address information and attach the one of available gateway. If many gateways are available in the network, sensor device IOT platform attaches only one of the gateways. In other side, sensor devices also transmit gateway info message for getting one of the available gateway address dynamically. Frequent transmission of advertise message, it impacts on gateway performance. In other side sensor node also frequently sending gateway info request to gateway, impacts on sensor node power and bandwidth.

B. Will Topic and will message feature

Will Topic and Will messages are useful whenever sensor device is abnormally disconnected from the gateway. Subscribers subscribe Will topic with MQTT-Server. Whenever sensor devices abnormally disconnected with the Gateway, MQTT Server delivers the will message to the Subscribed will topic for appropriate action. Whole sequence process as shown in **Fig-04**.

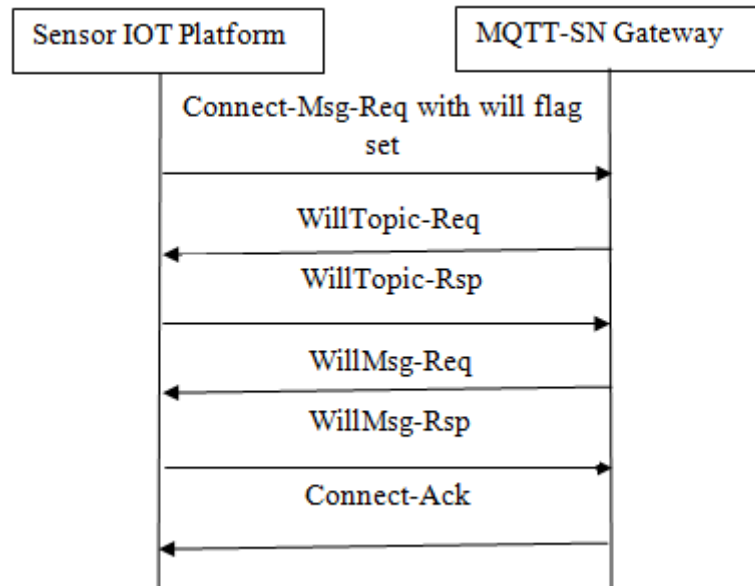


Fig 04: Will Topic and Message Sequence

C. MQTT-SN Registration procedure

In the MQTT Protocol Architecture, topic name length is two bytes, it takes a length of topic name string up to 65535, it is too long topic name for MQTT-SN protocol. It is not affordable for the power and bandwidth constrained sensor devices. To resolve this issue, MQTT-SN protocol provides registration procedure. Whenever Sensor device initiate connect req, registration request to the gateway, gateway responds with short topic id for that long topic name. Sensor node uses short topic id for publish procedure. Registration procedure as shown in **Fig-05**

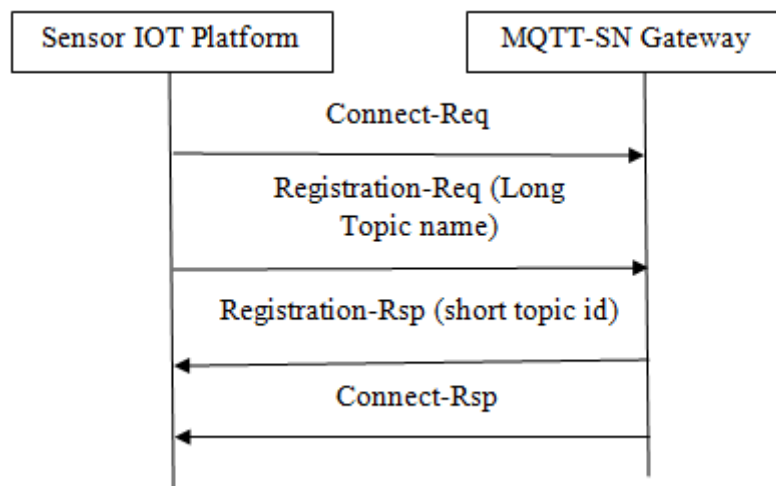


Fig 05: Registration sequence

D. Sleeping Client procedure

In MQTT-SN protocol sleeping client procedure is useful, when no data is being sent to Gateway, Sensor device goes into sleep state and informs the Gateway status of the sensor device. Due to this power is saved in the sensor device.

E. Publish and Subscribe procedures are remaining same as MQTT protocol, but short topic id is used for publishing messages instead of long topic name.

F. IOT protocol stack

IOT Protocol stacks as shown in Fig-06 for the Sensor device, Gateway and MQTT-Server.

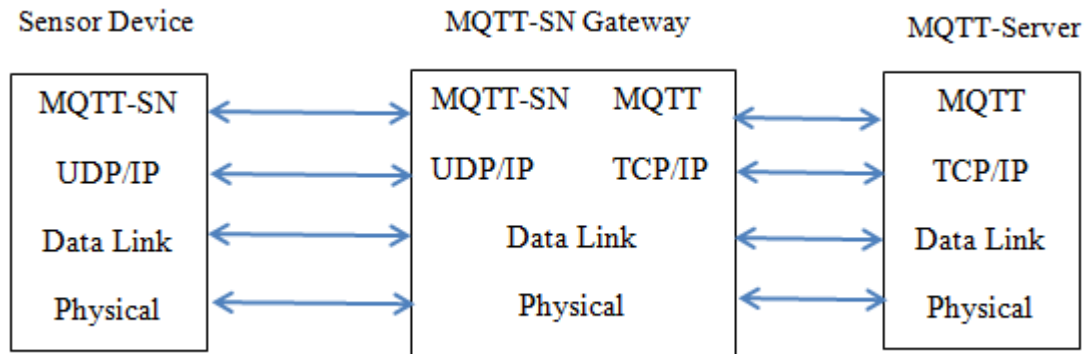


Fig -06: IOT Protocol stack

G. MQTT-SN architecture

In the MQTT-SN protocol three types of components are available 1.MQTT-SN Clients 2.MQTT-SN Gateways, 3.MQTT-SN Forwarder and MQTT-Server.MQTT-SN architecture as shown in Fig-07

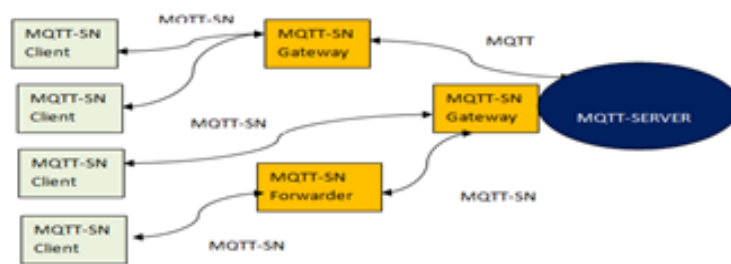


Fig 07:MQTT-SN Architecture

MQTT-SN Gateways are divided into transparent gateway, hybrid gateway and aggregating gateway.

Transparent Gateway: In this structure, each MQTT-SN client connects with MQTT-Server separately.

This is simple implementation in Gateway. But Number of MQTT-SN clients increases ,more number of MQTT-Connections are required in the gateway.

Hybrid Gateway: In this structure, some of the MQTT-SN clients are grouped and for this group only one MQTT-Connection initiated by the gateway. Moderate complexity is involved in the Gateway.

Aggregating Gateway: In this structure all MQTT-SN clients are grouped into one group and gateway initiate only one of MQTT-Connection with MQTT server. But more complexity involved in the gateway implementation.

H. MQTT-SN QOS table

In MQTT-SN protocol, application designer can choose right QoSmodel as per application requirement dynamically. Different typesof QOS models are as shown in Table-01

QOS Model	Description
QOS-0	The message is delivered at most once, or it is not delivered at all
QOS-1	The message is always delivered at least once. If the sender does not receive an acknowledgment, the message is sent again with the DUP flag set until an acknowledgment is received.
QOS-2	Publish payload messages are delivered exactly once
QOS-(-1 or 3)	Initial connection need not be required, directly publish the message using predefined or short topic ids.

Table-01: MQTT-SN QOS models

I. MQTT-SN Topic Management

Topic is a logical addressing entity in MQTT-SN and MQTT protocols. Subscriber can subscribe required topics for particular type data. Whenever MQTT Server received data on the topic publishes the data to the subscribed topics of subscribers.

Topic name is any time of string in the MQTT. But it is not standardized, application can chose desired topic name as per requirement.

For ex Topic Name: /Hall/Room1/Temperature, :/Hall/Room1/Humidity

Wild card topic also supports

For example: /Hall/Room1/# → from the /Home/Room1/ all types of data received from /Hall/Room1/
But in the lossy network and frequent transmission of data, a long topic consumes bandwidth. To resolve this issue MQTT-SN Client can send register message with long topic name to the gateway, gate returns the short topic id.

J. MQTT-SN QOS message flows

In MQTT-SN protocol, three message sequences are available based on the QoS model

A. QOS-0 Message Sequence flow

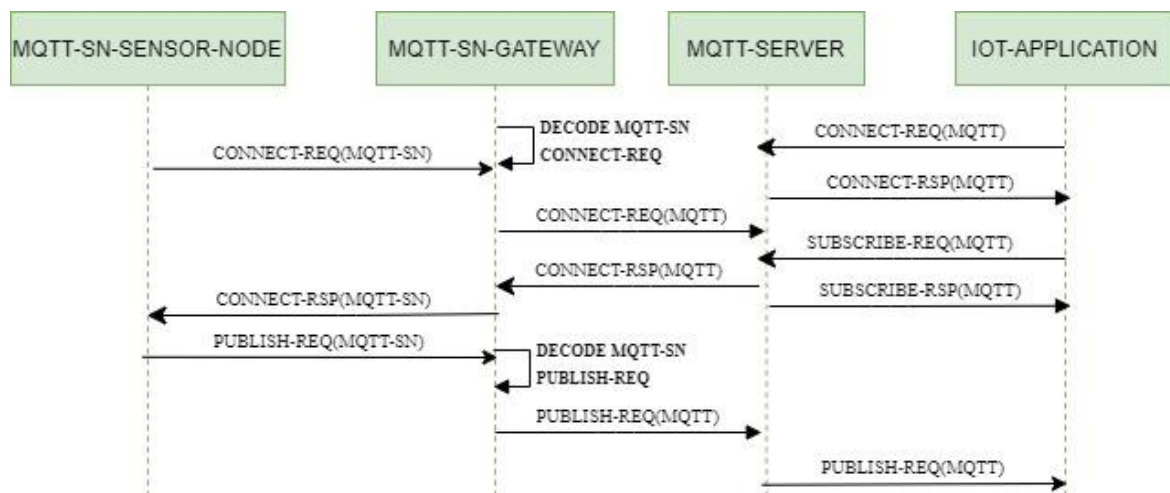


Fig-8 QOS-0 Message Sequence flow

B. QOS-1 Message Sequence flow

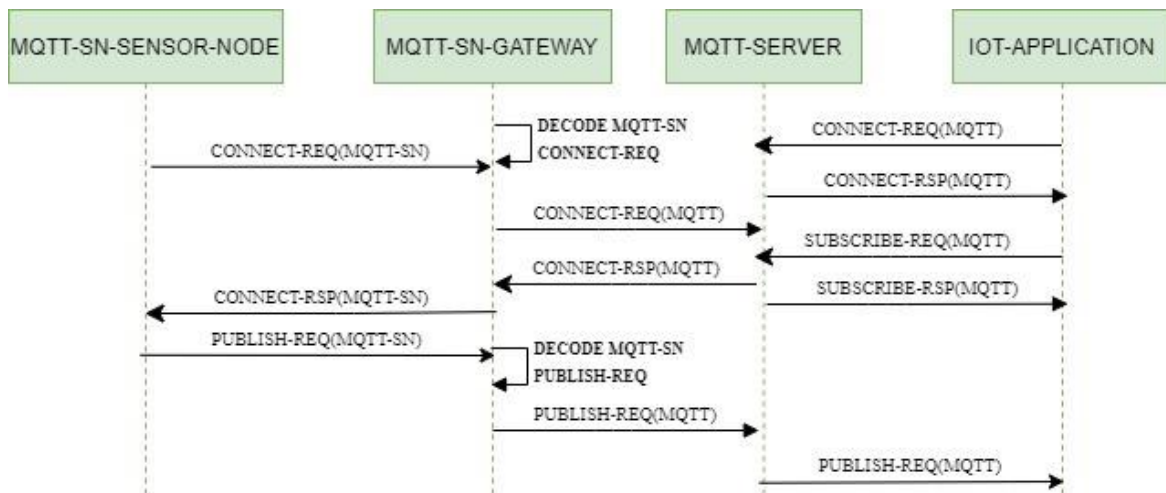


Fig 09: QOS-1 Message sequence flow

C. QOS-2 Message sequence flow

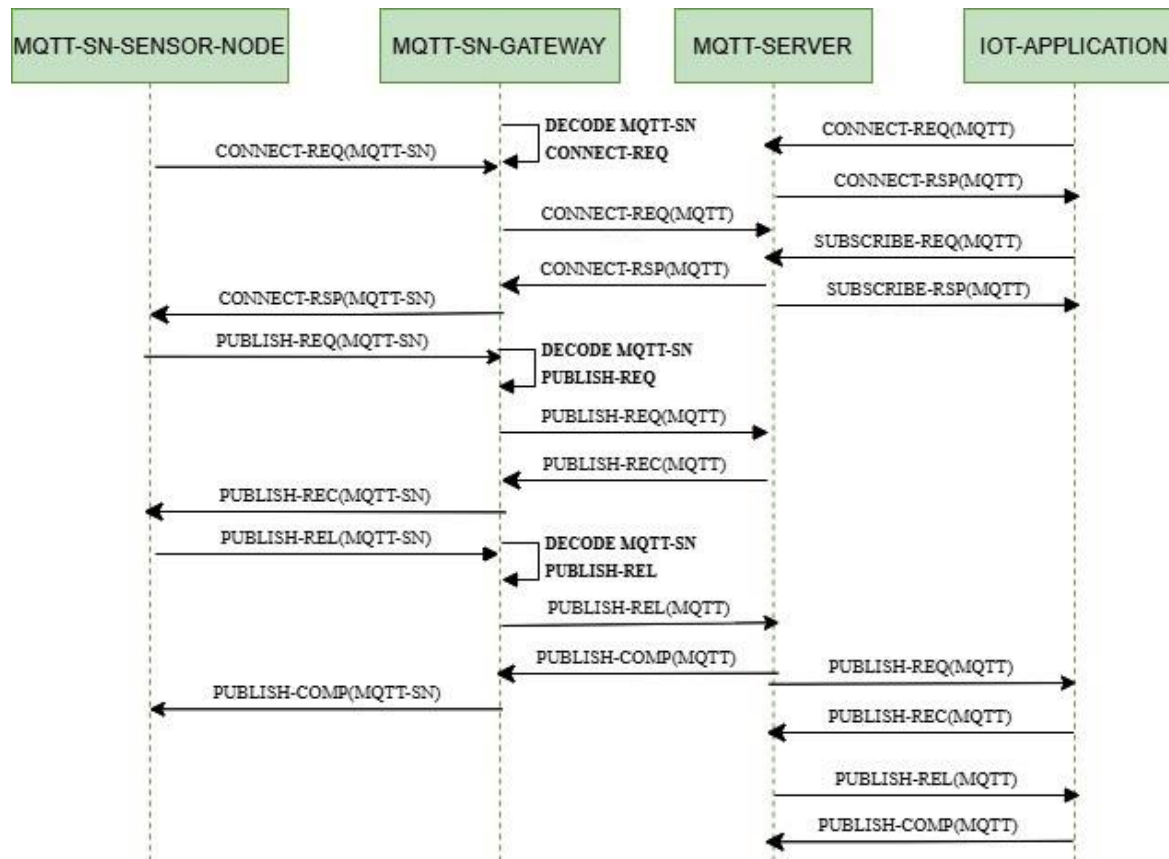


Fig 10:QOS-2 Message sequence flow

IV. IOT APPLICATION DEVELOPMENT PROCESS

MQTT Subscriber/MQTT-SN subscriber subscribe to the required sensor data through the topic name. MQTT Server receives the sensor data to that particular topic, MQTT server publishes the

sensor data to the subscribed topics. MQTT Subscriber receives the sensors data, decode it and store the sensor data in the SQL lite data base for further processing. MQTT Subscriber Application development process diagram as shown in **Fig-11**

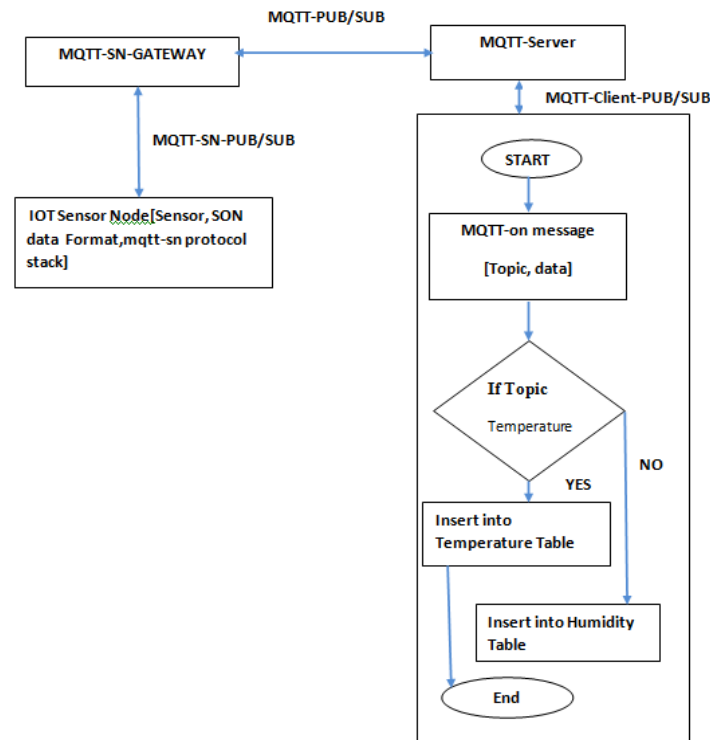


Fig 11: MQTT-Subscriber Application IOT Diagram

V. EXPERIMENTAL SETUP HARDWARE AND MESSAGE OVERHEAD, END TO END DELAY ANALYSIS

A. Practical Sensor Platform

DHT22 Sensor device connected to the ESP8286 low cost IOT platform .MQTT-SN stack software designed and developed to the Arduinio platform. Sensor Node software flow diagram mentioned in [4] by the same authors.

B. Gateway software deployment

Gateway MQTT-SN protocol software developed and deployed in the Raspberry Pi+B Hardware.

C. MQTT Server

MOSQUITTO MQTT server deployed in the desktop computing system and also cloud MQTT Server used for this project.

D. Simulation of MQTT-SN clients

C and Python based MQTT-SN clients, Android MQTT, MQTT mosquito publisher and subscriber frame work used for this simulation.

E. Message overhead analysis for MQTT-SN, MQTT and COAP

A. MQTT

A complete MQTT message publishing procedure involves the following steps/traffic exchange:

- Since the MQTT protocol relies on TCP, a new TCP connection must be established at the beginning of each MQTT session. This is a 3-way handshake involving both ends (packets #1-#3). In other words, within each MQTT session an underlying TCP connection is present.
- TCP connection establishment is followed by an MQTT connection establishment procedure, involving:
 - an MQTT connect message sent by the client (#4) and the respective TCP ACK message sent by the broker's TCP end-point (#5)
 - an MQTT connect ACK message sent by the MQTT broker (#6) and the respective TCP ACK message sent by the client's TCP end-point (#7)
- MQTT connection establishment is followed by the MQTT PUB procedure, which includes the MQTT communication credentials, the topic and the IoT payload information (#8)
- MQTT message publication is followed by the release of the MQTT connection, through the MQTT Disconnect Message sent by the client's MQTT end-point (#9).
- Following the FIN/ACK flags included in the TCP part of the MQTT disconnect command; the underlying TCP connection is released through a bi-directional TCP messages exchange (#10-#12).

In Table-02 shown the exact overhead per message, including the MQTT, TCP and IP overheads. Notice that the payload in our experiments (the [{"bn":"testdev-","n":"temp","u":"C","v":20.0}] string message) is 47-bytes long.

Procedure	MQTT OVERHEAD (Bytes)	TCPOVERHEAD (Bytes)	IPOVERHEAD (Bytes)	Total Overhead (Bytes)
TCP connection establishment (#1-#3)	-	40+40+32=112	3×20=60	172
MQTT connection establishment (#4)	115	32	20	167
MQTT connection establishment TCP ACK (#5)	–	32	20	52
MQTT connection ACK (#6)	4	32	20	56
MQTT connection ACK TCP ACK (#7)	–	32	20	52
MQTT message publication (#8)	144	32	20	196
MQTT disconnection (#9)	2	32	20	54
MQTT disconnection (#9)	2	32	20	54
Total overhead	265	400	240	905

Table-02-MQTT Message Overhead Calculation

In total, 905 bytes are consumed for a single 47 bytes payload transmission, i.e. almost 95% of bytes Exchanged are non-payload related. Assuming a 500KB NB-IoT data plan, this stands for roughly 550 messages per month or equivalently up to 18 messages per day, i.e. less than hourly status update.

B. CoAP protocol

Since CoAP is based on the UDP/IP transport protocol, there is no need for connection establishment/release and TCP acknowledgements. Hence, the CoAP POST command is encapsulated in a single UDP/IP message (packet #1). In the capture we also view a CoAP ACK message which is used to notify the client about successful message transmission. This is an optional feature of CoAP, so such a 2-way communication is not mandatory. The following table shows the overhead for each layer

Procedure	COAP OVERHEAD (Bytes)	UDP OVERHEAD (Bytes)	IPOVERHEAD (Bytes)	Total Overhead (Bytes)
CoAP POST (#1)	183	8	20	211
CoAP ACK (#2) (*optional)	6*	8*	20*	34*
Total overhead without Confirmed CoAP	183	8	20	211
Total overhead with Confirmed CoAP	189	16	40	245

Table-3-CoAP Message over head calculation

In total, for non-confirmed COAP, 211 bytes are consumed for a single 47 bytes payload transmission. Hence the overhead is significantly reduced compared to MQTT. The overall message length has been reduced more than 4 times. Assuming a 500 KB NB-IoT data plan, this stands for roughly 3,370 messages per month or equivalently up to 80 messages per day, i.e. approximately 3 status updates per hour.

C. MQTT-SN protocol

MQTT-SN stands “in-between” MQTT and CoAP, since it borrows the 2-way communication nature of the TCP-based MQTT protocol, but at the same time uses UDP/IP as the underlying transport mechanism. So, we expect some overhead savings due UDP usage.

- Recall that in standard MQTT, each time the client needs to publish something, the full topic name should be included in each message. This could be a long name, consuming a significant amount of bytes. In our example, the topic name is channels/8f3de729-6a42-48d5-a266-20db9c7bda35/messages/244629b1-389e-4b33-8a82-/ which is 92-bytes long. MQTT-SN introduces “topic registration”, where the long topic name could be mapped to a 2-byte integer, and afterwards this 2-byte field could be used in each publish message, instead of the whole string representation.
- In standard MQTT when the client is put to deep sleep (for energy consumption purposes) the one end of the TCP connection fails and the whole session is broken. Hence, after the node wakes up, the session needs to be restored from scratch. MQTT-SN introduces the asleep mode, during which the connection stays active. To achieve this, in MQTT-SN, the client does not establish an end-to-end TCP connection with the MQTT broker. Instead, an intermediate entity called the MQTT-SN Gateway, is responsible for translating UDP packets arriving from/destined to the client to MQTT packets destined to/arriving from the MQTT broker.

Procedure	MQTT OVERHEAD (Bytes)	UDP OVERHEAD (Bytes)	IPOVERHEAD (Bytes)	Total Overhead (Bytes)
MQTT-SN connection establishment (#1)	19	8	20	47
MQTT-SN connection ACK (#2)	3	8	20	31
MQTT-SN Topic Registration (* needed only once) (#3)	98	8	20	126
MQTT-SN Topic Registration ACK (* needed only once) (#4)	7	8	20	35
MQTT-SN Message Publication (#5)	54	8	20	82
MQTT-SN Disconnect to Sleep Mode (#6)	4	8	20	32
MQTT-SN Disconnect to Sleep Mode ACK (#7)	2	8	20	30
Total overhead	187	56	140	383

Table-4-MQTT-SN Message (with registration) over head calculation

Second scenario

Procedure	MQTT OVERHEAD (Bytes)	UDP OVERHEAD (Bytes)	IPOVERHEAD (Bytes)	Total Overhead (Bytes)
MQTT-SN (re-)connection establishment (#1)	19	8	20	47
MQTT-SN (re-)connection ACK (#2)	3	8	20	31
MQTT-SN Message Publication (#3)	54	8	20	82
MQTT-SN Disconnect to Sleep Mode (#4)	4	8	20	32
MQTT-SN Disconnect to Sleep Mode	2	8	20	30

ACK (#5)				
Total overhead	82	40	100	222

Table-5-MQTT-SN Message (with out registration) over head calculation

In total for MQTT-SN and assuming an IoT payload of 47 bytes, the overhead for the case where topic registration is also included is 383 bytes, whereas for the case where registration has already been performed is only 222 bytes, significantly lower than MQTT and similar to COAP.

D. The following table summarizes the key findings, under the following assumptions

- IoT Payload: 47 bytes
- NB-IoT monthly data plan: 500 KB

Protocol	OVERHEAD (BYTES)	TOTAL ALLOWED MESSAGES PER DAY
MQTT	905	18
MQTT-SN First Connection	383	43
MQTT-SN Communication after first connection	222	75
CoAP	211	80
MQTT	905	18

Table-06- Message overhead analysis for MQTT-SN,COAP,MQTT

E. QoS impact on performance of MQTT-SN protocol

There is a simple rule when considering performance impact of QoS. It is: “The higher the QoS, the lower the performance.” Let us evaluate performance corresponding with higher QoS. Suppose the time taken for sending a PUBLISH message is P_t . If QoS is used, the total time taken to transfer N number of messages will be $N P_t$. Now in case of QoS 1, the PUBACK message (that is reply for the PUBLISH message) will flow from server to client. This is a 2-byte message and might take a lot less time than P_t , hence call it m_t . So the time taken for transferring n messages will be $N(P_t + m_t)$. And for QoS 2, the PUBREC, PUBREL and PUBCOMP messages would be flowing. Hence the n number of messages would take approximately $N(P_t + 3m_t)$. So if 10 messages need to be transferred from client to server and P_t is 1 second and m_t is 0.4 seconds, a QoS 0 message would take $10 \times 1 = 10$ seconds, QoS 1 message would take $10(1 + 0.4)$ which is 14 seconds and QoS 2 message would take 22 seconds.

F. MQTT-SN Message percentage Losses

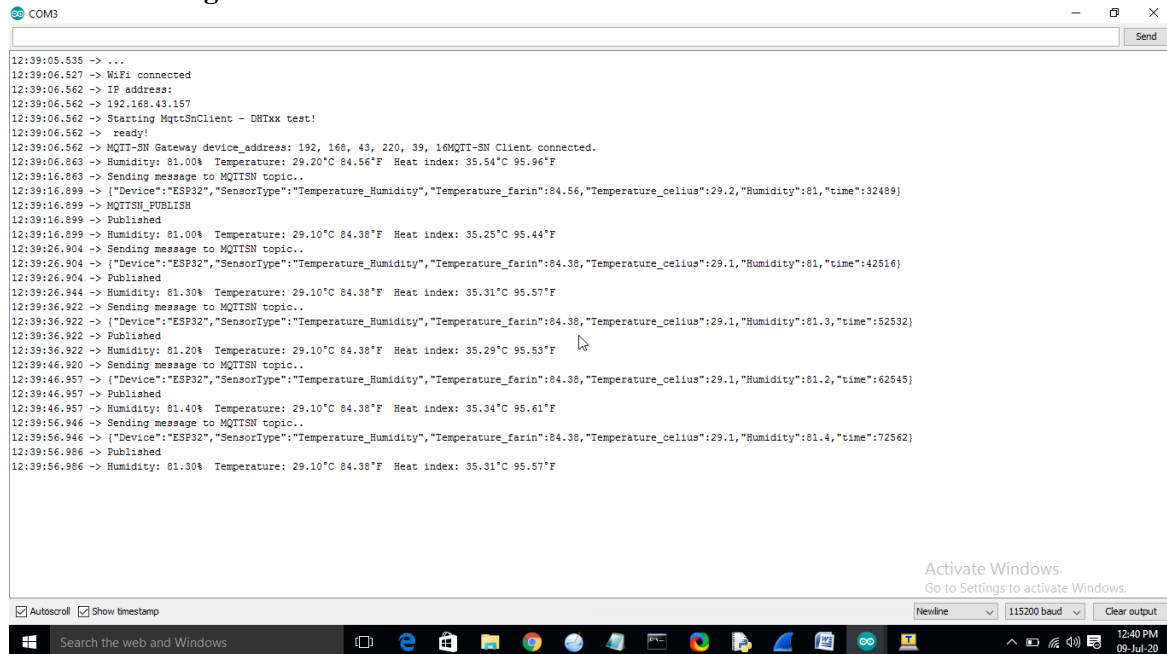
Message payload in bytes to published to the MQTT-Server using Python MQTT-SN script	Source to destination message loss calculation at QOS-Level-0 (%)	Source to destination message loss calculation at QOS-Level-1(%)	Source to destination message loss calculation at QOS-Level-2(%)
1000	1.00	0.24	.0.18
2000	1.40	0.41	0.2
3000	1.60	0.60	0.22
4000	1.80	0.78	0.24

Table-07 Message percentage losses

VI. PRACTICAL SIMULATION RESULTS

A. Case-01: End to End IOT Application Logs (Mosquito MQTTServer deployed in the Local computer)

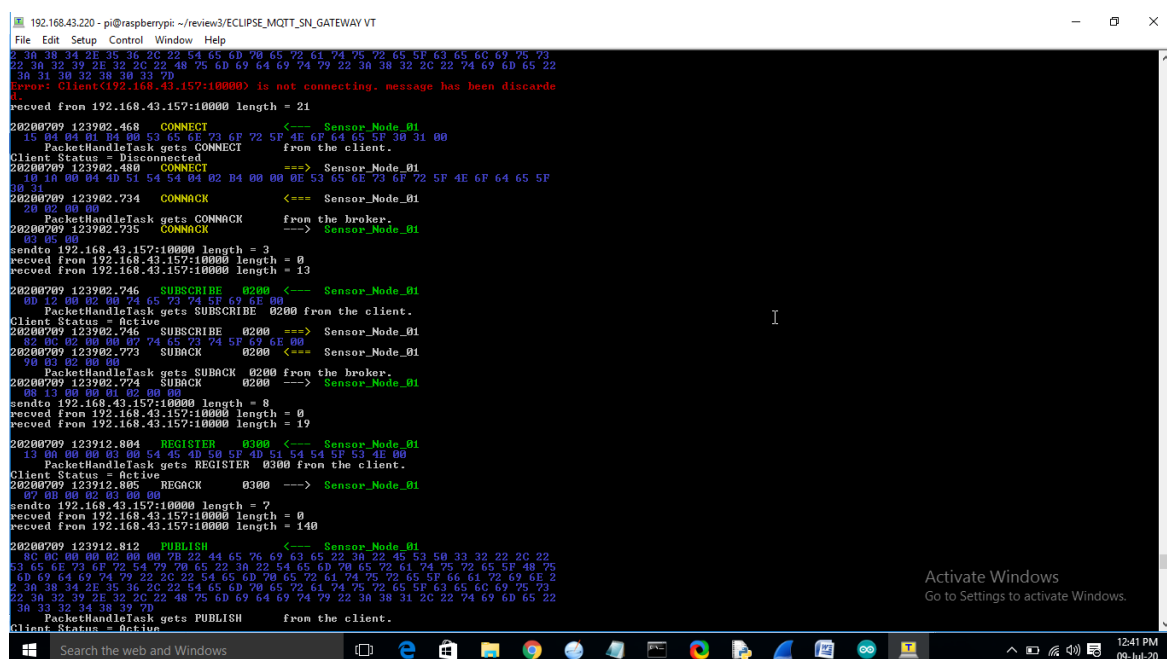
1.SensorNode Logs:



```
12:39:05.535 -> ...
12:39:06.527 -> WiFi connected
12:39:06.562 -> IP address: -> 192.168.43.157
12:39:06.562 -> Starting MqttSnClient - DHTxx test!
12:39:06.562 -> ready!
12:39:06.562 -> MQTT-SN Gateway device_address: 192, 168, 43, 220, 39, 16MQTT-SN Client connected.
12:39:06.563 -> Humidity: 81.00% Temperature: 29.20°C 84.56°F Heat index: 35.54°C 95.96°F
12:39:16.863 -> Sending message to MQTTSN topic..
12:39:16.899 -> [{"Device":"ESP32","SensorType":"Temperature_Humidity","Temperature_farin":84.56,"Temperature_celsius":29.2,"Humidity":81,"time":32489}
12:39:16.899 -> MQTTSN_PUBLISH
12:39:16.899 -> Published
12:39:16.899 -> Humidity: 81.00% Temperature: 29.10°C 84.38°F Heat index: 35.25°C 95.44°F
12:39:26.904 -> Sending message to MQTTSN topic..
12:39:26.904 -> [{"Device":"ESP32","SensorType":"Temperature_Humidity","Temperature_farin":84.38,"Temperature_celsius":29.1,"Humidity":81,"time":42516}
12:39:26.904 -> Published
12:39:26.944 -> Humidity: 81.30% Temperature: 29.10°C 84.38°F Heat index: 35.31°C 95.57°F
12:39:36.922 -> Sending message to MQTTSN topic..
12:39:36.922 -> [{"Device":"ESP32","SensorType":"Temperature_Humidity","Temperature_farin":84.38,"Temperature_celsius":29.1,"Humidity":81.3,"time":52532}
12:39:36.922 -> Published
12:39:36.922 -> Humidity: 81.20% Temperature: 29.10°C 84.38°F Heat index: 35.29°C 95.53°F
12:39:46.920 -> Sending message to MQTTSN topic..
12:39:46.957 -> [{"Device":"ESP32","SensorType":"Temperature_Humidity","Temperature_farin":84.38,"Temperature_celsius":29.1,"Humidity":81.2,"time":62545}
12:39:46.957 -> Published
12:39:46.957 -> Humidity: 81.40% Temperature: 29.10°C 84.38°F Heat index: 35.34°C 95.61°F
12:39:56.946 -> Sending message to MQTTSN topic..
12:39:56.946 -> [{"Device":"ESP32","SensorType":"Temperature_Humidity","Temperature_farin":84.38,"Temperature_celsius":29.1,"Humidity":81.4,"time":72562}
12:39:56.986 -> Published
12:39:56.986 -> Humidity: 81.30% Temperature: 29.10°C 84.38°F Heat index: 35.31°C 95.57°F
```

Fig: 12-Sensor Node Logs

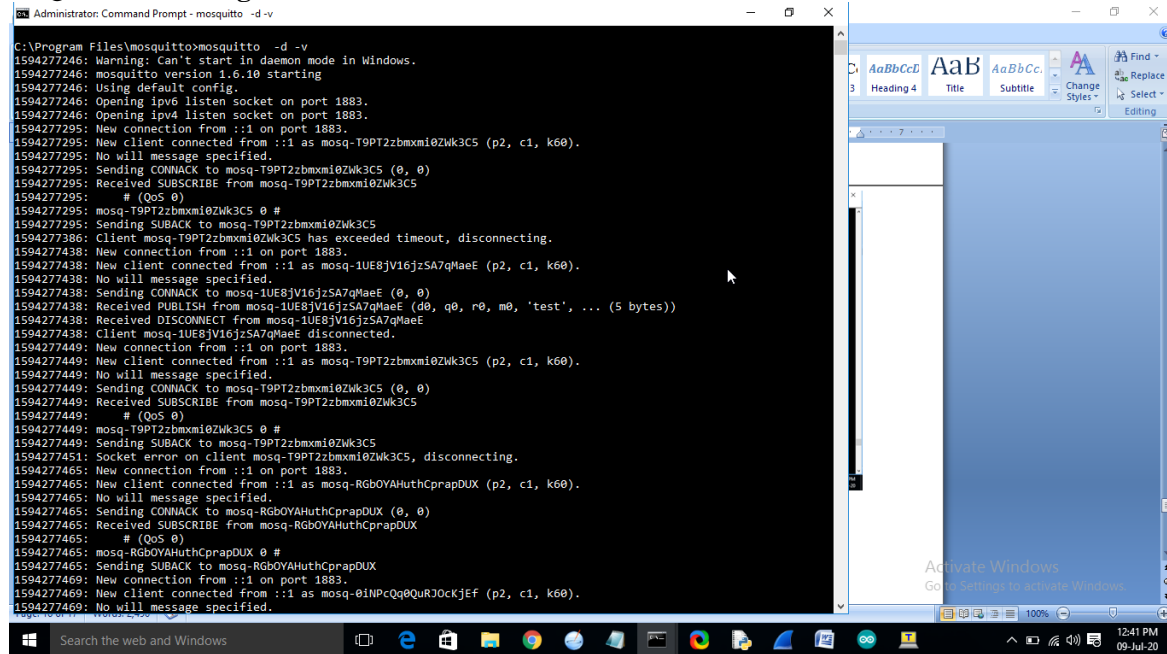
2.Gateway Logs:



```
192.168.43.220 - pi@raspberrypi: ~/review3/ECLIPSE_MQTT_SN_GATEWAY VT
File Edit Setup Control Window Help
2 38 30 34 2E 35 46 20 22 54 65 6D 70 65 72 61 74 75 72 65 5F 63 65 6C 69 75 73
22 3A 32 39 2E 32 2C 22 48 75 6D 69 64 69 74 79 22 38 38 32 2C 22 74 69 6D 65 22
38 31 30 32 38 32 38 32 38 32 38 32 38 32 38 32 38 32 38 32 38 32 38 32 38 32
Error: Client (192.168.43.157:10000) is not connecting, message has been discarded.
received from 192.168.43.157:10000 length = 21
20200709 123902.468 CONNECT <== Sensor_Node_01
15 04 04 01 24 00 53 65 6E 73 6F 72 5F 4E 6F 64 65 5F 30 31 00
PacketHandleTask gets CONNECT from the client.
Client Status = Disconnected
20200709 123902.480 CONNECT ==> Sensor_Node_01
10 10 00 04 4D 51 54 54 04 02 D4 00 00 0E 53 65 6E 73 6F 72 5F 4E 6F 64 65 5F
38 31
20200709 123902.734 CONNACK <== Sensor_Node_01
20 02 00 00
PacketHandleTask gets CONNACK from the broker.
20200709 123902.735 CONNACK ==> Sensor_Node_01
03 05 00
sendto 192.168.43.157:10000 length = 3
received from 192.168.43.157:10000 length = 0
received from 192.168.43.157:10000 length = 13
20200709 123902.746 SUBSCRIBE 0200 <== Sensor_Node_01
00 12 00 02 00 74 65 74 74 5F 69 4E 00
PacketHandleTask gets SUBSCRIBE 0200 from the client.
Client Status = Active
20200709 123902.746 SUBSCRIBE 0200 ==> Sensor_Node_01
82 02 02 00 00 07 74 65 74 74 5F 69 6E 00
20200709 123902.773 SUBACK 0200 <== Sensor_Node_01
70 03 02 00 00
PacketHandleTask gets SUBACK 0200 from the broker.
20200709 123902.774 SUBACK 0200 ==> Sensor_Node_01
08 13 00 00 01 02 00 00
sendto 192.168.43.157:10000 length = 8
received from 192.168.43.157:10000 length = 0
received from 192.168.43.157:10000 length = 19
20200709 123912.804 REGISTER 0300 <== Sensor_Node_01
13 00 00 00 03 00 54 45 4D 50 5F 4D 51 54 54 5F 63 4E 00
PacketHandleTask gets REGISTER 0300 from the client.
Client Status = Active
20200709 123912.805 REGACK 0300 ==> Sensor_Node_01
07 0B 00 02 03 00 00
sendto 192.168.43.157:10000 length = 7
received from 192.168.43.157:10000 length = 0
received from 192.168.43.157:10000 length = 140
20200709 123912.812 PUBLISH <== Sensor_Node_01
8C 0C 00 00 02 00 00 7B 22 44 65 75 69 63 65 22 3A 22 45 53 80 33 32 22 2C 22
53 65 6E 73 6F 72 54 79 79 65 22 3A 22 54 65 6D 70 65 72 61 74 75 72 65 5F 63 65 6C 69 75 73
6D 69 64 69 74 79 22 2C 22 54 65 6D 70 65 72 61 74 75 72 65 5F 66 61 72 69 6E 2
38 38 34 2E 35 46 20 22 54 65 6D 70 65 72 61 74 75 72 65 5F 63 65 6C 69 75 73
22 3A 32 39 2E 32 2C 22 48 75 6D 69 64 69 74 79 22 38 38 31 2C 22 74 69 6D 65 22
38 33 32 34 38 39 7D
PacketHandleTask gets PUBLISH from the client.
Client Status = Active
```

Fig:13-MQTT-SN Gateway Logs

3.MQTT-Server Logs



```
Administrator: Command Prompt - mosquitto -d -v
C:\Program Files\mosquitto>mosquitto -d -v
1594277246: Warning: Can't start in daemon mode in Windows.
1594277246: mosquitto version 1.6.10 starting
1594277246: Using default config.
1594277246: Opening ipv6 listen socket on port 1883.
1594277246: Opening ipv4 listen socket on port 1883.
1594277295: New connection from ::1 on port 1883.
1594277295: Client mosq-T9PT2zbxmxi0Zwk3C5 (p2, c1, k60).
1594277295: No will message specified.
1594277295: Sending CONNACK to mosq-T9PT2zbxmxi0Zwk3C5 (0, 0)
1594277295: Received SUBSCRIBE from mosq-T9PT2zbxmxi0Zwk3C5
1594277295: # (QoS 0)
1594277295: mosq-T9PT2zbxmxi0Zwk3C5 0 #
1594277295: Sending SUBACK to mosq-T9PT2zbxmxi0Zwk3C5
1594277386: Client mosq-T9PT2zbxmxi0Zwk3C5 has exceeded timeout, disconnecting.
1594277438: New connection from ::1 on port 1883.
1594277438: Client mosq-1UE8jV16jzSA7qMaeE (p2, c1, k60).
1594277438: No will message specified.
1594277438: Sending CONNACK to mosq-1UE8jV16jzSA7qMaeE (0, 0)
1594277438: Received DISCONNECT from mosq-1UE8jV16jzSA7qMaeE (d0, q0, r0, m0, 'test', ... (5 bytes))
1594277438: Client mosq-1UE8jV16jzSA7qMaeE disconnected.
1594277449: New connection from ::1 on port 1883.
1594277449: Client mosq-T9PT2zbxmxi0Zwk3C5 (p2, c1, k60).
1594277449: No will message specified.
1594277449: Sending CONNACK to mosq-T9PT2zbxmxi0Zwk3C5 (0, 0)
1594277449: Received SUBSCRIBE from mosq-T9PT2zbxmxi0Zwk3C5
1594277449: # (QoS 0)
1594277449: mosq-T9PT2zbxmxi0Zwk3C5 0 #
1594277449: Sending SUBACK to mosq-T9PT2zbxmxi0Zwk3C5
1594277451: Socket error on client mosq-T9PT2zbxmxi0Zwk3C5, disconnecting.
1594277465: New connection from ::1 on port 1883.
1594277465: Client mosq-RGbOYAHuthCprapDUX (p2, c1, k60).
1594277465: No will message specified.
1594277465: Sending CONNACK to mosq-RGbOYAHuthCprapDUX (0, 0)
1594277465: Received SUBSCRIBE from mosq-RGbOYAHuthCprapDUX
1594277465: # (QoS 0)
1594277465: mosq-RGbOYAHuthCprapDUX 0 #
1594277465: Sending SUBACK to mosq-RGbOYAHuthCprapDUX
1594277469: New connection from ::1 on port 1883.
1594277469: Client mosq-0iNpcQq0QuRJOckJfEf (p2, c1, k60).
1594277469: No will message specified.
```

Fig.14-MQTT Server Logs

4.IOT Application Logs

IOT Application is starting

Connected with result code 0

Subscribing topic

MQTT Data Received...

MQTT Topic: TEMP_MQTT_SN

data Received type <class 'str'>

data Received

```
{"Device":"ESP32","SensorType":"Temperature_Humidity","Temperature_farin":84.2,"Temperature_celsius":29,"Humidity":79.4,"time":32117}
```

Enter DHT22_Temp_Data_Handler

SensorID :ESP32

SensorType :Temperature_Humidity

Temperature_farin : 84.2

Temperature_celsius : 29.0

Humidity : 79.4

Data_and_Time : 09-Jul-2020 12:45:38:926518

MQTT Data Received...

MQTT Topic: TEMP_MQTT_SN

data Received type <class 'str'>

data Received

```
{"Device":"ESP32","SensorType":"Temperature_Humidity","Temperature_farin":84.2,"Temperature_celsius":29,"Humidity":81.2,"time":42147}
```

Enter DHT22_Temp_Data_Handler

SensorID :ESP32

SensorType :Temperature_Humidity

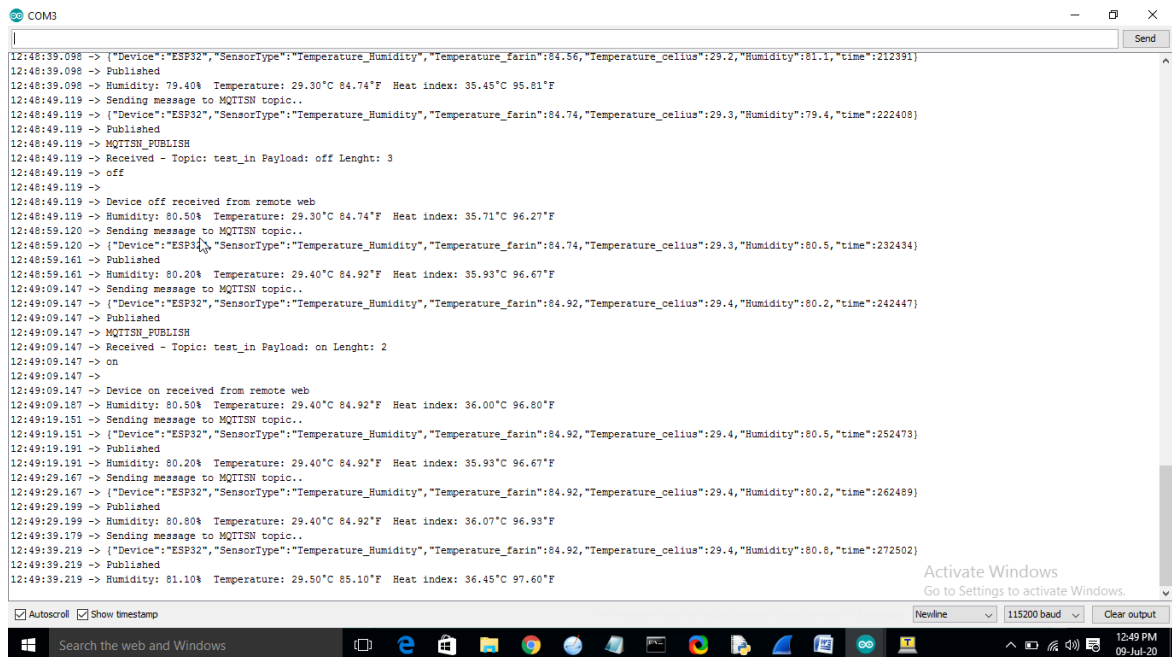
Temperature_farin : 84.2

Temperature_celsius : 29.0

Humidity : 81.2

Data_and_Time : 09-Jul-2020 12:45:48:952169

B. Case-02: End to End IOT Application Logs with Cloud MQTT Server




```
COM3
12:48:39.098 -> ["Device": "ESP32", "SensorType": "Temperature_Humidity", "Temperature_farin": 84.56, "Temperature_celsius": 29.2, "Humidity": 81.1, "time": 212391]
12:48:39.098 -> Published
12:48:39.098 -> Humidity: 79.40% Temperature: 29.30°C 84.74°F Heat index: 35.45°C 95.81°F
12:48:49.119 -> Sending message to MQTTSN topic..
12:48:49.119 -> ["Device": "ESP32", "SensorType": "Temperature_Humidity", "Temperature_farin": 84.74, "Temperature_celsius": 29.3, "Humidity": 79.4, "time": 222408]
12:48:49.119 -> Published
12:48:49.119 -> MQTTSN_PUBLISH
12:48:49.119 -> Received - Topic: test_in Payload: off Lenght: 3
12:48:49.119 -> off
12:48:49.119 ->
12:48:49.119 -> Device off received from remote web
12:48:49.119 -> Humidity: 80.50% Temperature: 29.30°C 84.74°F Heat index: 35.71°C 96.27°F
12:48:59.120 -> Sending message to MQTTSN topic..
12:48:59.120 -> ["Device": "ESP32", "SensorType": "Temperature_Humidity", "Temperature_farin": 84.74, "Temperature_celsius": 29.3, "Humidity": 80.5, "time": 232434]
12:48:59.161 -> Published
12:48:59.161 -> Humidity: 80.20% Temperature: 29.40°C 84.92°F Heat index: 35.93°C 96.67°F
12:48:59.161 -> Sending message to MQTTSN topic..
12:48:59.161 -> ["Device": "ESP32", "SensorType": "Temperature_Humidity", "Temperature_farin": 84.92, "Temperature_celsius": 29.4, "Humidity": 80.2, "time": 242447]
12:48:59.161 -> Published
12:48:59.161 -> MQTTSN_PUBLISH
12:48:59.161 -> Received - Topic: test_in Payload: on Lenght: 2
12:48:59.161 -> on
12:48:59.161 ->
12:48:59.161 -> Device on received from remote web
12:48:59.161 -> Humidity: 80.50% Temperature: 29.40°C 84.92°F Heat index: 36.00°C 96.80°F
12:48:59.161 -> Sending message to MQTTSN topic..
12:48:59.161 -> ["Device": "ESP32", "SensorType": "Temperature_Humidity", "Temperature_farin": 84.92, "Temperature_celsius": 29.4, "Humidity": 80.5, "time": 252473]
12:48:59.161 -> Published
12:48:59.161 -> Humidity: 80.20% Temperature: 29.40°C 84.92°F Heat index: 35.93°C 96.67°F
12:48:59.161 -> Sending message to MQTTSN topic..
12:48:59.161 -> ["Device": "ESP32", "SensorType": "Temperature_Humidity", "Temperature_farin": 84.92, "Temperature_celsius": 29.4, "Humidity": 80.2, "time": 262489]
12:48:59.161 -> Published
12:48:59.161 -> Humidity: 80.80% Temperature: 29.40°C 84.92°F Heat index: 36.07°C 96.93°F
12:48:59.161 -> Sending message to MQTTSN topic..
12:48:59.161 -> ["Device": "ESP32", "SensorType": "Temperature_Humidity", "Temperature_farin": 84.92, "Temperature_celsius": 29.4, "Humidity": 80.8, "time": 272502]
12:48:59.161 -> Published
12:48:59.161 -> Humidity: 81.10% Temperature: 29.50°C 85.10°F Heat index: 36.45°C 97.60°F
12:48:59.161 -> Published

Activate Windows
Go to Settings to activate Windows.
```

5. Sensor Node Logs

Fig:15-Sensor Node Logs(With Cloud MQTT Server)

6. Gateway Logs



```
192.168.43.220 - pi@raspberrypi: ~/review3/ECLIPSE_MQTT_SN_GATEWAY_VT
File Edit Setup Control Window Help
SensorN/W: UDP Multicast 225.1.1.1:1883 Gateway Port 10000
Broker: hairdresser.cloudmqtt.com : 15738, 8883
RootCApath: <null>
RootCAfile: <null>
CertKey: <null>
PrivateKey: <null>

received from 192.168.43.157:10000 length = 21
20200709 131910.410 CONNECT <-- Sensor_Node_01
PacketHandleTask gets CONNECT from the client.
Client Status = Disconnected
20200709 131911.419 CONNECT ==> Sensor_Node_01
08 72 65 60 79 61 62 70 65 00 0C 4F 43 51 55 38 62 6F 78 46 6C 4A 63
20200709 131911.739 CONNACK <-- Sensor_Node_01
PacketHandleTask gets CONNACK from the broker.
20200709 131911.740 CONNACK ==> Sensor_Node_01
03 05 00
sendto 192.168.43.157:10000 length = 3
received from 192.168.43.157:10000 length = 0
received from 192.168.43.157:10000 length = 13
20200709 131911.758 SUBSCRIBE 0200 <-- Sensor_Node_01
PacketHandleTask gets SUBSCRIBE 0200 from the client.
Client Status = Active
20200709 131911.759 SUBSCRIBE 0200 ==> Sensor_Node_01
20200709 131911.955 SUBACK 0200 <-- Sensor_Node_01
PacketHandleTask gets SUBACK 0200 from the broker.
20200709 131911.956 SUBACK 0200 ==> Sensor_Node_01
08 13 00 00 01 02 00 00
sendto 192.168.43.157:10000 length = 8
received from 192.168.43.157:10000 length = 0
received from 192.168.43.157:10000 length = 19
20200709 131921.984 REGISTER 0300 <-- Sensor_Node_01
PacketHandleTask gets REGISTER 0300 from the client.
Client Status = Active
20200709 131921.985 REGACK 0300 ==> Sensor_Node_01
07 08 00 02 03 00 00
sendto 192.168.43.157:10000 length = 7
received from 192.168.43.157:10000 length = 0
received from 192.168.43.157:10000 length = 142
20200709 131921.997 PUBLISH <-- Sensor_Node_01
73 6F 72 54 79 70 65 22 3A 22 54 65 6D 70 65 72 61 74 75 72 65 5F 48 75 6D 69 64 69 74 79 22 2C 22 54 65 6D 70 65 72 61 74 75 72 65 5F 48 75 6D 69 64 69 74 79 22 3A 32 39 2E 31 2C 22 48 75 6D 69 64 69 74 79 22 3A 37 38 2E 35 2C 22 74 69 6D 65 22 3A 31 36 31 34 32 70
PacketHandleTask gets PUBLISH from the client.
Client Status = Active
20200709 131921.997 PUBLISH ==> Sensor_Node_01
30 95 01 00 0C 54 45 4D 50 5F 4D 51 54 54 5F 53 4E 7B 22 44 65 76 69 63 65 22 3A 22 45
53 50 33 32 22 2C 22 53 65 6E 73 6F 72 54 79 70 65 22 3A 22 54 65 6D 70 65 72 61 74 75 72 65 5F 48 75 6D 69 64 69 74 79 22 2C 22 54 65 6D 70 65 72 61 74 75 72 65 5F 48 75 6D 69 64 69 74 79 22 3A 32 39 2E 31 2C 22 48 75 6D 69 64 69 74 79 22 3A 37 38 2E 35 2C 22 74 69 6D 65 22 3A 31 36 31 34 32 70
61 72 65 6D 22 3A 38 34 2E 33 30 2C 22 54 65 6D 70 65 72 61 74 75 72 65 5F 48 75 6D 69 64 69 74 79 22 3A 32 39 2E 31 2C 22 48 75 6D 69 64 69 74 79 22 3A 37 38 2E 35 2C 22 74 69 6D 65 22 3A 31 36 31 34 32 70
69 6D 65 22 3A 31 36 31 34 32 70
received from 192.168.43.157:10000 length = 21
20200709 132109.488 CONNECT <-- Sensor_Node_01
PacketHandleTask gets CONNECT from the client.
Client Status = Disconnected
```

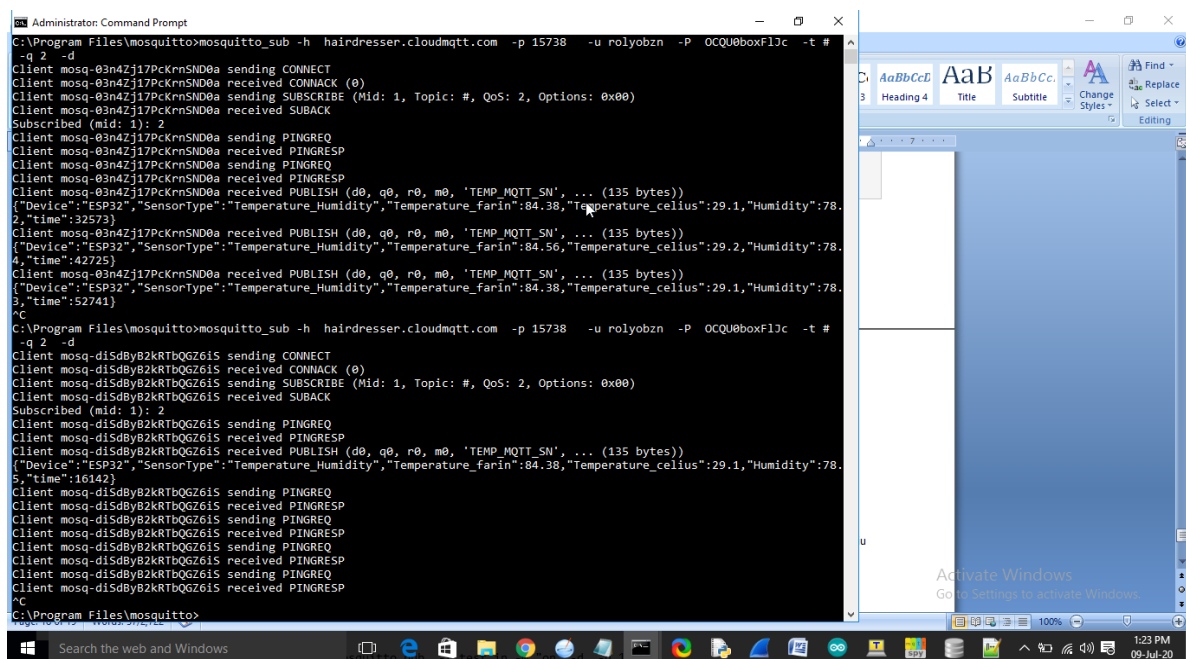
Fig:16-Gateway Node Logs(With Cloud MQTT Server)

7.Cloud MQTT-Server Logs

```
2020-07-09 07:51:09: New connection from 223.228.98.199 on port 15738.
2020-07-09 07:51:09: New client connected from 223.228.98.199 as Sensor_Node_01 (c1, k46080, u'rolyobzn').
2020-07-09 07:51:09: No will message specified.
2020-07-09 07:51:09: Sending CONNACK to Sensor_Node_01 (0, 0)
2020-07-09 07:51:10: Received SUBSCRIBE from Sensor_Node_01
2020-07-09 07:51:10: test_in (QoS 0)
2020-07-09 07:51:10: Sending SUBACK to Sensor_Node_01
2020-07-09 07:51:17: Received PINGREQ from mosq-diSdByB2kRTbQGZ6iS
2020-07-09 07:51:17: Sending PINGRESP to mosq-diSdByB2kRTbQGZ6iS
2020-07-09 07:51:53: Socket error on client <unknown>, disconnecting.
2020-07-09 07:51:55: New client connected from 223.228.98.199 as cloudmqtt-ws-ui-0.3054179418283789 (c1, k60, u'rolyobzn').
```

Fig: 17-Cloud MQTT Server Logs (With Cloud MQTT Server)

8.IOT Application Logs



```
Administrator: Command Prompt
C:\Program Files\mosquitto>mosquitto_sub -h hairdresser.cloudmqtt.com -p 15738 -u rolyobzn -P OCQU8boxF13c -t #
-q 2 -d
Client mosq-03n4Zj17PcKrnSND0a sending CONNECT
Client mosq-03n4Zj17PcKrnSND0a received CONNACK (0)
Client mosq-03n4Zj17PcKrnSND0a sending SUBSCRIBE (Mid: 1, Topic: #, QoS: 2, Options: 0x00)
Client mosq-03n4Zj17PcKrnSND0a received SUBACK
Subscribed (mid: 1): 2
Client mosq-03n4Zj17PcKrnSND0a sending PINGREQ
Client mosq-03n4Zj17PcKrnSND0a received PINGRESP
Client mosq-03n4Zj17PcKrnSND0a sending PINGREQ
Client mosq-03n4Zj17PcKrnSND0a received PINGRESP
Client mosq-03n4Zj17PcKrnSND0a received PUBLISH (d0, q0, r0, m0, 'TEMP_MQTT_SN', ... (135 bytes))
{"Device": "ESP32", "SensorType": "Temperature_Humidity", "Temperature_farin": 84.38, "Temperature_celsius": 29.1, "Humidity": 78.2, "time": 32573}
Client mosq-03n4Zj17PcKrnSND0a received PUBLISH (d0, q0, r0, m0, 'TEMP_MQTT_SN', ... (135 bytes))
{"Device": "ESP32", "SensorType": "Temperature_Humidity", "Temperature_farin": 84.56, "Temperature_celsius": 29.2, "Humidity": 78.4, "time": 42725}
Client mosq-03n4Zj17PcKrnSND0a received PUBLISH (d0, q0, r0, m0, 'TEMP_MQTT_SN', ... (135 bytes))
{"Device": "ESP32", "SensorType": "Temperature_Humidity", "Temperature_farin": 84.38, "Temperature_celsius": 29.1, "Humidity": 78.3, "time": 52741}
^C
C:\Program Files\mosquitto>mosquitto_sub -h hairdresser.cloudmqtt.com -p 15738 -u rolyobzn -P OCQU8boxF13c -t #
-q 2 -d
Client mosq-diSdByB2kRTbQGZ6iS sending CONNECT
Client mosq-diSdByB2kRTbQGZ6iS received CONNACK (0)
Client mosq-diSdByB2kRTbQGZ6iS sending SUBSCRIBE (Mid: 1, Topic: #, QoS: 2, Options: 0x00)
Client mosq-diSdByB2kRTbQGZ6iS received SUBACK
Subscribed (mid: 1): 2
Client mosq-diSdByB2kRTbQGZ6iS sending PINGREQ
Client mosq-diSdByB2kRTbQGZ6iS received PINGRESP
Client mosq-diSdByB2kRTbQGZ6iS sending PINGREQ
Client mosq-diSdByB2kRTbQGZ6iS received PINGRESP
Client mosq-diSdByB2kRTbQGZ6iS sending PUBLISH (d0, q0, r0, m0, 'TEMP_MQTT_SN', ... (135 bytes))
{"Device": "ESP32", "SensorType": "Temperature_Humidity", "Temperature_farin": 84.38, "Temperature_celsius": 29.1, "Humidity": 78.5, "time": 116140}
Client mosq-diSdByB2kRTbQGZ6iS sending PINGREQ
Client mosq-diSdByB2kRTbQGZ6iS received PINGRESP
Client mosq-diSdByB2kRTbQGZ6iS sending PINGREQ
Client mosq-diSdByB2kRTbQGZ6iS received PINGRESP
Client mosq-diSdByB2kRTbQGZ6iS sending PINGREQ
Client mosq-diSdByB2kRTbQGZ6iS received PINGRESP
Client mosq-diSdByB2kRTbQGZ6iS sending PINGREQ
Client mosq-diSdByB2kRTbQGZ6iS received PINGRESP
^C
C:\Program Files\mosquitto>
```

Fig: 18-IOT Application Logs(With Cloud MQTT Server)

9.MQTT-SN-QOS-0,QOS-1,QOS-2 Wire shark Logs

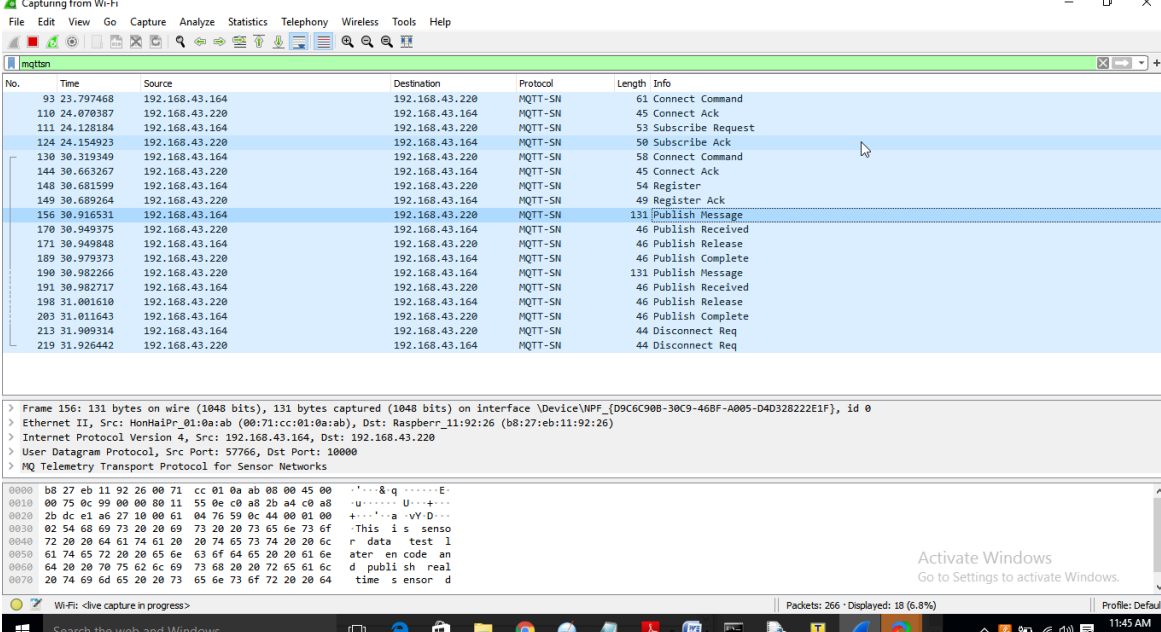


Figure 19 shows a Wireshark capture of MQTT-SN traffic. The packet list on the left shows a series of messages between 192.168.43.164 and 192.168.43.220. The selected packet (No. 156) is a Publish Message (MQTT-SN) with a length of 131 bytes. The packet details pane shows the structure of the MQTT-SN message, including the Topic Name (b8 27 eb 11 92 26 00 71 cc 01 0a ab 08 00 45 00) and the Payload (00 75 0c 99 00 00 00 11 55 0e c0 a8 2b a4 c0 a8). The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
93	23.797468	192.168.43.164	192.168.43.220	MQTT-SN	61	Connect Command
110	24.070387	192.168.43.220	192.168.43.164	MQTT-SN	45	Connect Ack
111	24.128184	192.168.43.164	192.168.43.220	MQTT-SN	53	Subscribe Request
124	24.154923	192.168.43.220	192.168.43.164	MQTT-SN	50	Subscribe Ack
130	30.319349	192.168.43.164	192.168.43.220	MQTT-SN	58	Connect Command
144	30.663267	192.168.43.220	192.168.43.164	MQTT-SN	45	Connect Ack
148	30.681599	192.168.43.164	192.168.43.220	MQTT-SN	54	Register
149	30.689264	192.168.43.220	192.168.43.164	MQTT-SN	49	Register Ack
156	30.918531	192.168.43.164	192.168.43.220	MQTT-SN	131	Publish Message
170	30.949375	192.168.43.220	192.168.43.164	MQTT-SN	46	Publish Received
171	30.949848	192.168.43.164	192.168.43.220	MQTT-SN	46	Publish Release
189	30.979373	192.168.43.220	192.168.43.164	MQTT-SN	46	Publish Complete
190	30.982266	192.168.43.164	192.168.43.220	MQTT-SN	131	Publish Message
191	30.982717	192.168.43.164	192.168.43.220	MQTT-SN	46	Publish Received
198	31.001610	192.168.43.220	192.168.43.164	MQTT-SN	46	Publish Release
203	31.011643	192.168.43.164	192.168.43.220	MQTT-SN	46	Publish Complete
213	31.909314	192.168.43.164	192.168.43.220	MQTT-SN	44	Disconnect Req
219	31.926442	192.168.43.220	192.168.43.164	MQTT-SN	44	Disconnect Req

Fig: 19-MQTT-SN-QOS-02-Wireshark Logs

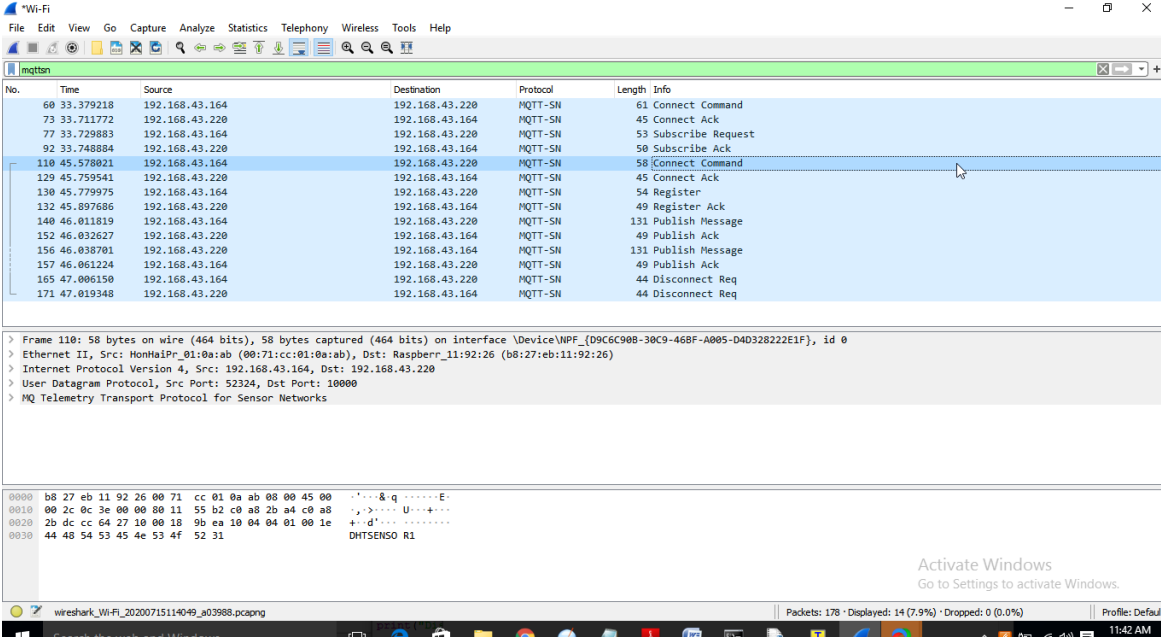


Figure 20 shows a Wireshark capture of MQTT-SN traffic. The packet list on the left shows a series of messages between 192.168.43.164 and 192.168.43.220. The selected packet (No. 110) is a Connect Command (MQTT-SN) with a length of 58 bytes. The packet details pane shows the structure of the MQTT-SN message, including the Topic Name (b8 27 eb 11 92 26 00 71 cc 01 0a ab 08 00 45 00) and the Payload (00 75 0c 99 00 00 00 11 55 0e c0 a8 2b a4 c0 a8). The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
60	33.379218	192.168.43.164	192.168.43.220	MQTT-SN	61	Connect Command
73	33.711772	192.168.43.220	192.168.43.164	MQTT-SN	45	Connect Ack
77	33.729883	192.168.43.164	192.168.43.220	MQTT-SN	53	Subscribe Request
92	33.748884	192.168.43.220	192.168.43.164	MQTT-SN	50	Subscribe Ack
110	45.578021	192.168.43.164	192.168.43.220	MQTT-SN	58	Connect Command
129	45.759541	192.168.43.220	192.168.43.164	MQTT-SN	45	Connect Ack
130	45.779975	192.168.43.164	192.168.43.220	MQTT-SN	54	Register
132	45.897686	192.168.43.220	192.168.43.164	MQTT-SN	49	Register Ack
140	46.011819	192.168.43.164	192.168.43.220	MQTT-SN	131	Publish Message
152	46.032627	192.168.43.220	192.168.43.164	MQTT-SN	49	Publish Ack
156	46.038781	192.168.43.220	192.168.43.164	MQTT-SN	131	Publish Message
157	46.061224	192.168.43.164	192.168.43.220	MQTT-SN	49	Publish Ack
165	47.006150	192.168.43.164	192.168.43.220	MQTT-SN	44	Disconnect Req
171	47.019348	192.168.43.220	192.168.43.164	MQTT-SN	44	Disconnect Req

Fig: 20-MQTT-SN-QOS-01-Wireshark Logs

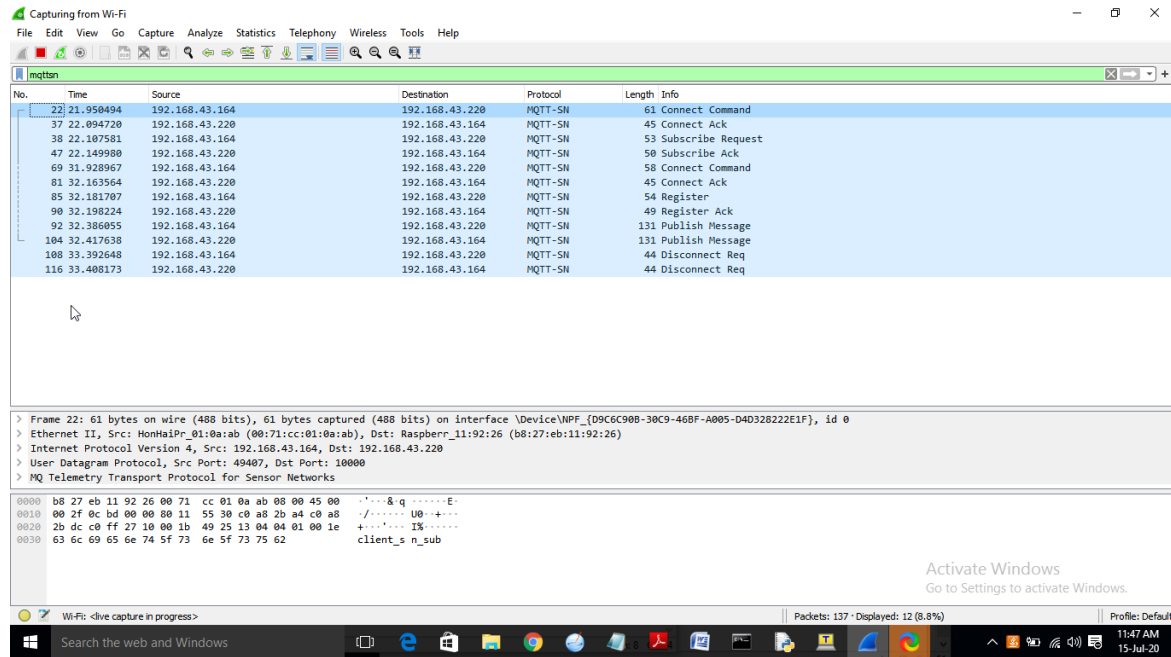


Fig: 20-MQTT-SN-QOS-0-Wireshark Logs

10.MQTT-Application Received Sensor Data

Practical temperature and humidity sensor data received as shown in Fig-21

id	Device	Sensor Type	Temperature_farin	Temperature_celsius	Humidity	Date_n Time
1	ESP32	Temperature_Humidity	84.56	29.2	51.2	09-July-2020 09:19:03:308728
2	ESP32	Temperature_Humidity	84.56	29.2	51.2	09-July-2020 09:19:13:316439
3	ESP32	Temperature_Humidity	84.56	29.2	51.2	09-July-2020 09:19:23:416277
4	ESP32	Temperature_Humidity	84.56	29.2	51.4	09-July-2020 09:19:33:460665
5	ESP32	Temperature_Humidity	84.56	29.2	51.5	09-July-2020 09:19:43:458445
6	ESP32	Temperature_Humidity	84.56	29.2	51.7	09-July-2020 09:19:53:467951
7	ESP32	Temperature_Humidity	84.56	29.2	51.9	09-July-2020 09:20:03:453599
8	ESP32	Temperature_Humidity	84.56	29.2	51.8	09-July-2020 09:20:13:470940
9	ESP32	Temperature_Humidity	84.56	29.2	51.9	09-July-2020 09:20:23:493006
10	ESP32	Temperature_Humidity	84.56	29.2	51.8	09-July-2020 09:20:33:522815

Fig: 21-MQTT-Application Received Sensor Data

11.End to End Delay versus Number of messages

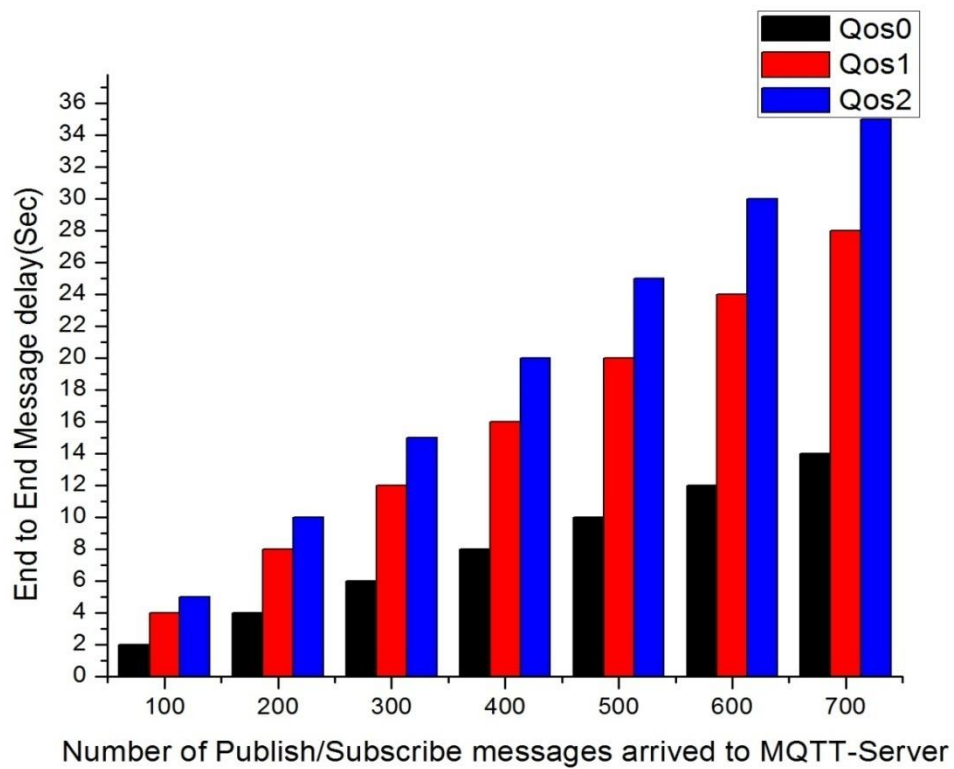


Fig: 20-Number of publisher/subscriber arrived to MQTT-Server

12.End to End Delay versus message size

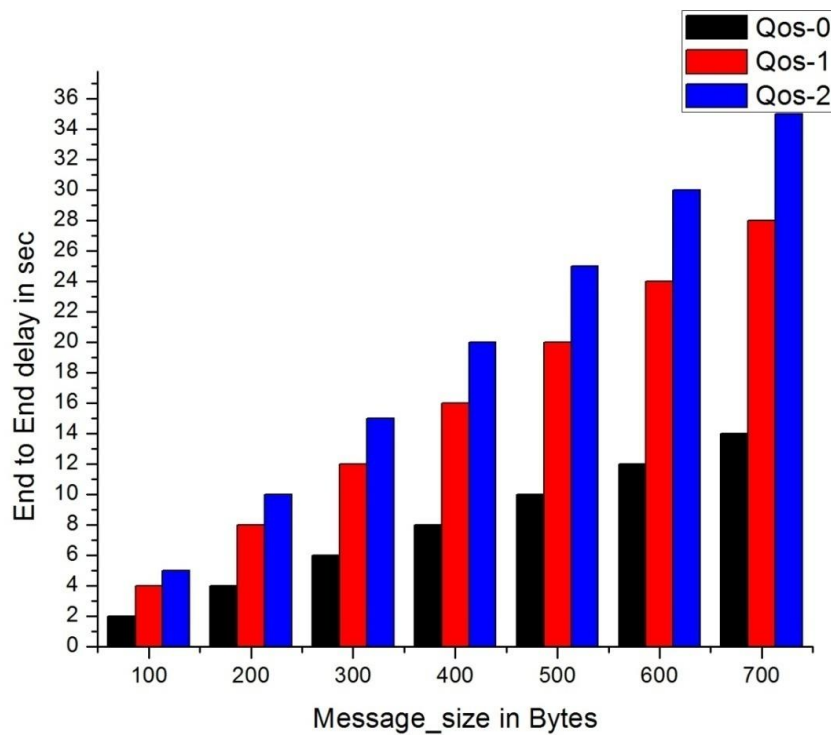


Fig-21 MQTT-SN Message Size in Bytes

13. Temperature Sensor data graph

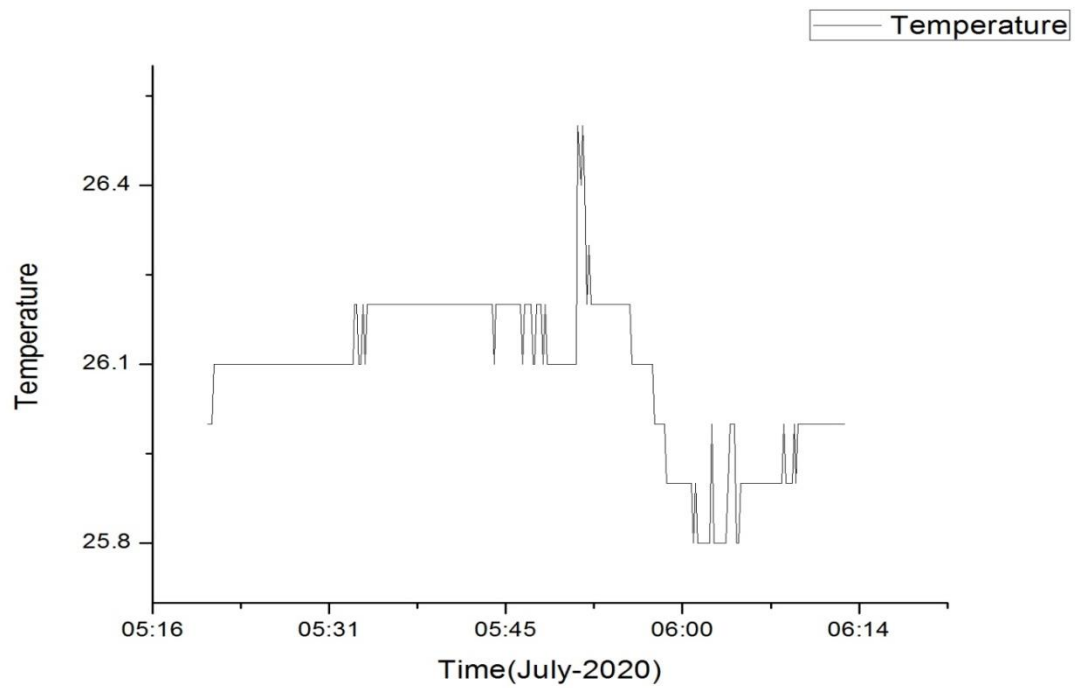


Fig-22-Temperature Sensor data

14. Humidity Sensor Data graph

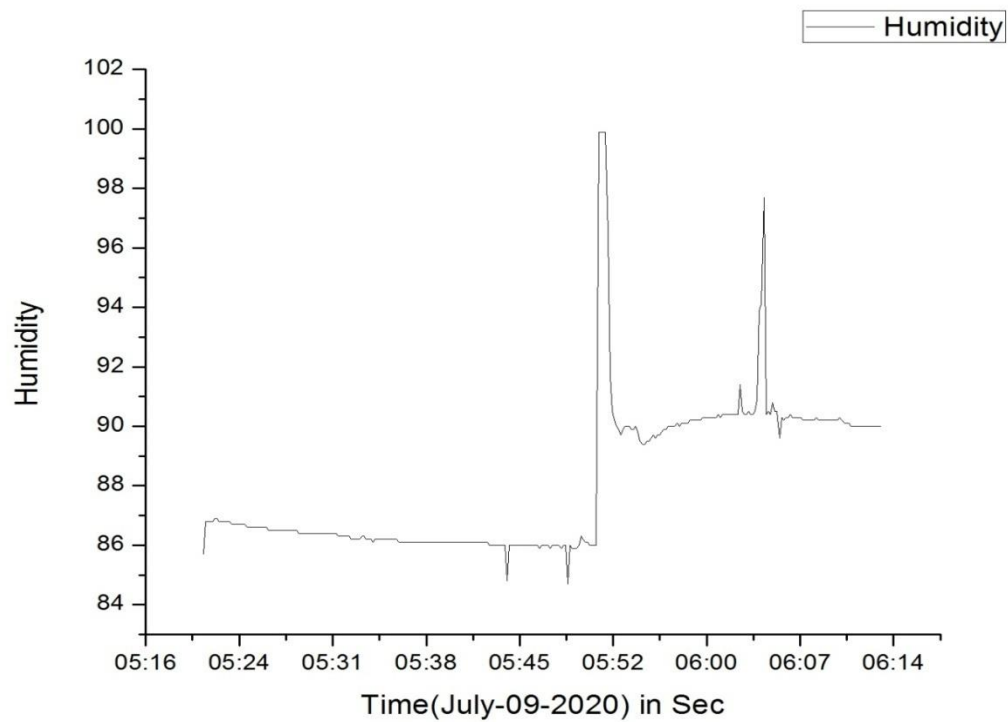


Fig-23-Humidity Sensor data

VII. CONCLUSION

In this paper we discussed end to end message delay calculations for different types MQTT-SN QoS, MQTT-SN gateway integrated with local mosquito MQTT Server and Cloud MQTT Server. Simple MQTT IOT application developed for receiving of Sensor data from the MQTT-Server. Infuture we will implement this setup and different QoS Models for the practical IOT applications.

REFERENCES

1. http://www.mqtt.org/new/wpcontent/uploads/2009/06/MQTT-SN_spec_v1.2.pdf
2. Naik, N., "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP", in 2017 IEEE International Systems Engineering Symposium, 2017.
3. Govindan, K., & Azad, A. P. "End-to-end service assurance in IoT MQTT-SN.", 12th Annual IEEE Consumer Communications and Networking Conference, 2015
4. Urs Hunkeler, I HongLinh Truong, Andy Stanford-Clark "MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks"- 3rd International Conferences on Communication Systems Software and Middleware and workshops, 2008.
5. Andre Gloria, Francisco Cercas, Nuno Souto - "Design and implementation of an IoT gateway to create smart environments", in 8th International Conference on Ambient Systems, Networks and Technologies, Elsevier, pp.568–575, 2017.
6. Guha Roy, D., Mahato, B., De, D., & Buyya, R. "Application-aware end-to-end delay and message loss estimation in Internet of Things (IoT) — MQTT-SN protocols". Future Generation Computer Systems, Vol 89, pp. 300–316, 2018
7. Tantitharanukul, N., Osathanunkul, K., Hantrakul, K., Pramokchon, P. & Khoenkaw, P, "MQTT-Topics Management System for sharing of Open Data." in International Conference on Digital Arts, Media and Technology, 2017
8. Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. "Internet of Things for Smart Cities.", IEEE Internet of Things Journal, Vol 1, pp. 22–32, 2014.
9. ESP8266 Arduino Core Documentation Release 2.4.0
10. <https://nodered.org/>
11. http://mqtt.org/documentation/MQTT_v3.1.1.pdf
12. F. Jerald, M. Anand, N. Deepika, "Design of an Industrial IOT Architecture Based on MQTT Protocol for End Device to Cloud Communication", in International Journal of Recent Technology and Engineering, Vol-7, ISSN: 2277-3878, 2019.
13. Gopi Krishna, P., Sreenivasa Ravi, K., Hari Kishore, K., Krishna Veni, K., N. Siva Rao, K., & D. Prasad, R. "Design and development of bi-directional IoT gateway using ZigBee and Wi-Fi technologies with MQTT protocol.", International Journal of Engineering & Technology, Vol-7, pp. 125-129, 2018.
14. Amaran, M. H. Noh, N. A. M., Rohmad, M. S., & Hashim, H. "A comparison of Lightweight Communication Protocols in Robotic Applications.", Procedia Computer Science, Vol 76, pp. 400–405, 2015
15. Guoqiang, S., Yanming, C., Chao, Z., & Yanxu, Z. "Design and Implementation of a Smart IoT Gateway." IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, 2013
16. <https://www.sqlite.org>
17. <https://www.python.org>
18. <http://www.steves-internet-guide.com/mqtt-sn>
19. <https://www.eclipse.org/paho/components/mqtt-sn-transparent-gateway/>

AUTHORS PROFILE



Mr. M.Obula Reddy received his B.Tech degree in Electronics and Communication from JNTU University, Hyderabad, A.P. M.Tech from NIT, Calicut, Kerala. He has 11 Years of Industry experience in Telecom and Data communication protocol Engg.7 Year of Teaching experience in Engineering for UG&PG courses.



J.B.Seventline received B.Tech degree in ECE from Bharathiar University, Coimbatore. TN.ME from Madurai kamaraj University, Madurai,TN and Ph.D in Radar Signal Processing from Andhra University, Visakhapatnam. She has 26 years of teaching experience and presently working as Professor of ECE, GITAM deemed to be University, Visakhapatnam. She has almost 50 technical papers published in reputed journals and conferences. Her research interests include Radar Signal Processing, Image Processing, VLSI Signal Processing and Internet of things application messaging protocols