This application note describes the accompanying example code in which Talaria TWO publishes a text message to a MQTT Broker and subscribes to a topic and displays the received message over Talaria TWO’s console. The application can be evaluated in secure and non-secure modes of MQTT operations.

# MQ Telemetry Transport Protocol

MQTT is a messaging protocol based on publish-subscribe pattern. It works on top of the TCP/IP protocol and is used in internet of things.

The publish-subscribe paradigm is event-driven, and messages are pushed to the clients. It requires an additional central point, called MQTT Broker, which takes care of dispatching all the messages between the senders and the rightful receivers.

When the clients publish messages to the broker, they include a topic into the message. For the broker, the topic acts as the routing information. Each client that wants to receive messages subscribes to a certain topic and the broker takes care of delivering all messages with the matching topic to the relevant client.

There is no requirement for the clients to know each other directly and the communication happens only over the topic. This pattern removes the dependencies of direct connectivity between the data producers and the data consumers and thus enables scalable solutions.

Apart from this, the protocol has the property of the client side requiring a small code footprint and less bandwidth, while the MQTT Broker can do the heavy lifting of receiving messages from thousands of clients concurrently, filtering and routing each message to the right client subscribed to the topic.

This makes the MQTT protocol ideal for resource constrained IoT devices, which need to be bandwidth-efficient and use little battery power.

# Relevant APIs

## MQTT APIs

### MQTTNetworkInit()

Initializes MQTT network object with socket read, write, and disconnect functions.

|  |
| --- |
| void MQTTNetworkInit(MQTTNetwork\* n) |

### MQTTNetworkInit\_Tls()

Initializes MQTT network object with socket read, write, and disconnect functions for TLS.

|  |
| --- |
| void MQTTNetworkInit\_Tls(MQTTNetwork\*); |

### MQTTNetworkConnect()

Opens a socket and tries to connect the MQTT network object to the network endpoint.

|  |
| --- |
| int MQTTNetworkConnect(MQTTNetwork\* n, char\* addr, int port) |

### MQTTNetworkConnect\_Tls()

Opens a socket and tries to connect the MQTT network object to the network endpoint over TLS.

|  |
| --- |
| int MQTTNetworkConnect\_Tls(MQTTNetwork \*n, char \*host, int port, ssl\_wrap\_cfg\_t \*cfg); |

### MQTTNetworkDisconnect()

Closes the socket and tries to disconnect the MQTT network object to the network endpoint.

|  |
| --- |
| void MQTTNetworkDisconnect(MQTTNetwork \*n) |

### MQTTNetworkDisconnect\_Tls()

Closes the socket over TLS and tries to disconnect the MQTT network object to the network endpoint.

|  |
| --- |
| void MQTTNetworkDisconnect\_Tls(MQTTNetwork\*); |

### MQTTClientInit()

Creates an MQTT client object.

|  |
| --- |
| void MQTTClientInit(MQTTClient\* client, MQTTNetwork\* network, unsigned int command\_timeout\_ms,unsigned char\* sendbuf, size\_t sendbuf\_size, unsigned char\* readbuf, size\_t readbuf\_size); |

### MQTTConnect()

Sends an MQTT connect packet down the network and waits for a CONNACK. The network object must be connected to the network endpoint before calling this.

|  |
| --- |
| int MQTTConnect(MQTTClient\* client, MQTTPacket\_connectData\* options); |

### MQTTDisconnect()

Sends an MQTT disconnect packet and closes the connection.

|  |
| --- |
| int MQTTDisconnect(MQTTClient\* client); |

### MQTTPublish()

Sends an MQTT publish packet and waits for all acks to complete.

|  |
| --- |
| int MQTTPublish(MQTTClient\* client, const char \*topic, MQTTMessage \*message); |

### MQTTSubscribe()

Sends an MQTT subscribe packet and waits for SUBACK before returning.

|  |
| --- |
| int MQTTSubscribe(MQTTClient\* client, const char\* topicFilter, enum QoS qos, MQTTMessageHandler messageHandler); |

### MQTTUnsubscribe()

Sends an MQTT unsubscribe packet and waits for UNSUBACK before returning.

|  |
| --- |
| int MQTTUnsubscribe(MQTTClient\* client, const char\* topicFilter); |

### MQTTYield()

MQTT goes to the background for the time (ms) to yield for.

|  |
| --- |
| int MQTTYield(MQTTClient\* client, int time); |

### MQTTNoPollInit()

Initializes MQTT without polling for incoming packets. This API blocks the thread until a message to the subscribed topic is received.

|  |
| --- |
| int MQTTNoPollInit(void); |

### MQTTNetworkInit\_Ws ()

Initializes the connection handle passed to the API. This API needs to be called if the intended MQTT connection is over Websocket.

|  |
| --- |
| void MQTTNetworkInit\_Ws(MQTTNetwork\* handle); |

### MQTTNetworkConnect\_Ws ()

Connects to Broker over Websocket. The connection can either be secured or non-secured.

|  |
| --- |
| int MQTTNetworkConnect\_Ws(MQTTNetwork\* n, websock\_config\_t \* ws\_cfg); |

# Application Flow

In this application, Talaria TWO is programmed to publish a message to the MQTT Broker running as a MQTT instance. Talaria TWO also subscribes to a topic and displays the received message on the console. The application supports a secured connection over SSL/TLS to the broker and can connect to the broker(s) with two different MQTT connections.

Following is the application flow:

1. Connect the device to a Wi-Fi network, whose SSID and passphrase are given as boot arguments while flashing the binary image.
2. Connect to the MQTT instance using the URL, port, username, and password of the cloud which are also given as boot arguments.

For a secured connection, the URL, port, username, password, transport mode, path of CA certificate, path of Client certificate, and path of Client key are provided as boot arguments.

When the mqtt\_no\_poll boot argument is set to 1, there is no polling mechanism involved to yield the incoming messages. It is recommended to set the mqtt\_no\_poll to 1 to reduce power consumption.

1. Subscribe to a topic to receive the messages.
2. Publish a message for another topic every 2 seconds.
3. The published messages can be seen on the subscriber’s console window.
4. The messages subscribed to the topic can be seen on the Download tool’s console.

# Code Walkthrough

## Mounting the Filesystem

The filesystem is mounted to be able to access the certificates required to perform the SSL/TLS handshake.

|  |
| --- |
| rval = utils\_mount\_rootfs();  if(0 != rval) {  os\_printf("Error: Mounting rootfs\n");  return -1;  } |

## Reading the Boot Argument

The following bootargs are to be passed to the application while programming the Talaria TWO with mqtt.elf:

1. SSID and Passphrase of Wi-Fi Network
2. URL, Port, Username ,Password of the MQTT server and the required number of MQTT connections.
3. Transport mode(TCP, TLS or WS), Path of CA certificate, Path of Client certificate, Path of Client key.

|  |
| --- |
| bargs.ssid = os\_get\_boot\_arg\_str("ssid");  bargs.passphrase = os\_get\_boot\_arg\_str("passphrase");  bargs.cloud\_url = os\_get\_boot\_arg\_str("cloud\_url") ;  bargs.cloud\_port = os\_get\_boot\_arg\_int("cloud\_port", 1883);  bargs.cloud\_usr\_name = os\_get\_boot\_arg\_str("cloud\_usr\_name");  bargs.cloud\_usr\_psw = os\_get\_boot\_arg\_str("cloud\_usr\_psw");  bargs.ca\_cert = os\_get\_boot\_arg\_str("ca\_cert");  bargs.client\_cert = os\_get\_boot\_arg\_str("client\_cert");  bargs.client\_key = os\_get\_boot\_arg\_str("client\_key");  bargs.pub\_qos = os\_get\_boot\_arg\_int("pub\_qos", 1);  bargs.sub\_qos = os\_get\_boot\_arg\_int("sub\_qos", 0);  bargs.transport\_mode = os\_get\_boot\_arg\_int("transport\_mode", 0); |

num\_conn boot argument allows the user to configure the number of connections. The application is configured with the following default value num\_conn = 1. By using this boot argument, user can create a maximum of two connections.

|  |
| --- |
| bargs.num\_conn = os\_get\_boot\_arg\_int("num\_conn", 1); |

mqtt\_no\_poll boot argument is set to 1 by default to avoid polling through boot argument. If the application is enabled with no poll (mqtt\_no\_poll = 1), the application does not perform polling and the MQTT thread blocks until the message is received. Set this flag to reduce the power consumption.

|  |
| --- |
| bargs.mqtt\_no\_poll = os\_get\_boot\_arg\_int("mqtt\_no\_poll", 0); |

websock\_url boot argument allows the user to configure the websocket URL to connect the Talaria TWO websocket client.

|  |
| --- |
| bargs.websock\_url = os\_get\_boot\_arg\_str("websock\_url"); |

## Selecting the Transport Mode

1. Set transport\_mode=0 for unencrypted, unauthenticated connection.
2. Set transport\_mode=1 for server authentication. This is an encrypted connection and will require the server certificate.
3. Set transport\_mode=2 for mutual authentication. This mode also supports client authentication if the server intends to do so. This is an encrypted connection and will require a server certificate and the client certificate.
4. Set transport\_mode=3 for a secured connection without connection verification. This connection is encrypted and does not need certificates.
5. Set transport\_mode=4 for unencrypted, unauthenticated websocket connection.
6. Set transport\_mode=5 for websocket server authentication. This is an encrypted connection and will require the server certificate. For websocket server authentication requires a server certificate.
7. Set transport\_mode=6 for secured websocket no certificate verification.

|  |
| --- |
| if(bargs->transport\_mode == TCP){  m->transport = APP\_MQTT\_TM\_TCP;  }else if(bargs->transport\_mode == TLS){  m->transport = APP\_MQTT\_TM\_TLS;  }else if(bargs->transport\_mode == TLS\_CLIENT\_VERFIY){  m->transport = APP\_MQTT\_TM\_TLS;/\*With Client Authentication if the  Server forces it\*/  }else if(bargs->transport\_mode == TLS\_NO\_CERT\_VERIFY){  m->transport = APP\_MQTT\_TM\_TLS\_NO\_CERT\_VERIFY;  }else if(bargs->transport\_mode == WEBSOCK){  m->transport = APP\_MQTT\_TM\_WEBSOCK;  }else if(bargs->transport\_mode == SECURED\_WEBSOCK){  m->transport = APP\_MQTT\_TM\_SECURED\_WEBSOCK;  }else if(bargs->transport\_mode == SECURED\_WEBSOCK\_NO\_CERT\_VERIFY){  m->transport = APP\_MQTT\_TM\_SECURED\_WEBSOCK\_NO\_CERT\_VERIFY;  } |

## Connecting to a Wi-Fi Network

The wcm\_create() API starts creating the Wi-Fi network interface. wifi\_connect\_to\_network() API connects to the Wi-Fi network. This API waits indefinitely for the Wi-Fi connection as the argument WCM\_CONN\_WAIT\_INFINATE is passed.

|  |
| --- |
| h= wcm\_create(NULL);  rval = wifi\_connect\_to\_network(&h, WCM\_CONN\_WAIT\_INFINATE, &wcm\_connected);  if(rval < 0) {  os\_printf("\nError: Unable to connect to network\n");  return 0;  }  if(wcm\_connected != true) {  os\_printf("\nCouldn't Connect to network");  wcm\_disconnect(h);  return -1;  } |

If the Wi-Fi connection is successful, wcm\_connected flag is set to True.

## Initializing MQTT Client

On successfully establishing a Wi-Fi connection, an application thread is created to handle the MQTT connection, publish and subscribe operations. It initiates either the polling based MQTT operation or non-polling-based operation depending upon the value of the bootarg: mqtt\_no\_poll.

The app\_mqtt\_params\_set(&bargs, &mqtt\_param\_1) function copies the MQTT configuration data received from the boot arguments variable bargs to mqtt\_param\_1. The app\_auto\_generate\_params(&mqtt\_param\_1, 1) function generates a unique MQTT client ID based on Talaria TWO’s MAC ID. The subscribe and publish topics are also generated based on the generated MQTT client ID.

|  |
| --- |
| if(bargs.mqtt\_no\_poll){  /\*NO Polling for incoming packets\*/  MQTTNoPollInit();  }  app\_mqtt\_params\_set(&bargs, &mqtt\_param\_1);  app\_auto\_generate\_params(&mqtt\_param\_1, 1);    mqtt\_param\_2 = mqtt\_param\_1;  app\_auto\_generate\_params(&mqtt\_param\_2, 2); |

This application initiates two connection to the broker(s). Hence, app\_auto\_generate\_params() function is called again to generate another MQTT client ID, publish and subscribe topics.

app\_mqtt\_params\_set() API copies the MQTT configuration data from bootargs, selects the appropriate transport mode and provides pointers to the certificates and the client key.

|  |
| --- |
| m->cloud\_url = bargs->cloud\_url;  m->cloud\_port = bargs->cloud\_port;  m->cloud\_usr\_name = bargs->cloud\_usr\_name;  m->cloud\_usr\_psw = bargs->cloud\_usr\_psw;  m->ca\_cert = bargs->ca\_cert;  m->client\_cert = bargs->client\_cert;  m->client\_key = bargs->client\_key;  m->pub\_qos = bargs->pub\_qos;  m->sub\_qos = bargs->sub\_qos;  m->websock\_url = bargs->websock\_url;  m->mqtt\_cmd\_timeout = bargs->mqtt\_cmd\_timeout;    if(bargs->transport\_mode == TCP){  m->transport = APP\_MQTT\_TM\_TCP;  }else if(bargs->transport\_mode == TLS){  m->transport = APP\_MQTT\_TM\_TLS;  }else if(bargs->transport\_mode == TLS\_CLIENT\_VERFIY){  m->transport = APP\_MQTT\_TM\_TLS;/\*With Client Authentication if the  Server forces it\*/  }else if(bargs->transport\_mode == TLS\_NO\_CERT\_VERIFY){  m->transport = APP\_MQTT\_TM\_TLS\_NO\_CERT\_VERIFY;  }else if(bargs->transport\_mode == WEBSOCK){  m->transport = APP\_MQTT\_TM\_WEBSOCK;  }else if(bargs->transport\_mode == SECURED\_WEBSOCK){  m->transport = APP\_MQTT\_TM\_SECURED\_WEBSOCK;  }else if(bargs->transport\_mode == SECURED\_WEBSOCK\_NO\_CERT\_VERIFY){  m->transport = APP\_MQTT\_TM\_SECURED\_WEBSOCK\_NO\_CERT\_VERIFY;  }  m->cfg.auth\_mode = SSL\_WRAP\_VERIFY\_NONE;  if(m->transport == APP\_MQTT\_TM\_TLS || m->transport == APP\_MQTT\_TM\_SECURED\_WEBSOCK){  if(m->ca\_cert !=NULL){  /\*CA certificate\*/  m->cfg.ca\_cert.buf = utils\_file\_get(m->ca\_cert, &m->cfg.ca\_cert.len);  if(m->cfg.ca\_cert.buf == NULL){  os\_printf("Provide a valid path for the CA certificate-1\r\n");  return -2;  }  }else{  os\_printf("Provide a valid path for the CA certificate\r\n");  return -2;  }  m->cfg.auth\_mode = SSL\_WRAP\_VERIFY\_REQUIRED;  }  if(m->client\_cert != NULL){  /\*Client certificate\*/  m->cfg.client\_cert.buf = utils\_file\_get(m->client\_cert, &m->cfg.client\_cert.len);  if(m->cfg.client\_cert.buf == NULL && bargs->transport\_mode == TLS\_CLIENT\_VERFIY){  os\_printf("Provide a valid path for client certificate\r\n");  return -2;  }  }else if(bargs->transport\_mode == TLS\_CLIENT\_VERFIY){  os\_printf("Provide a valid path for client certificate\r\n");  return -2;  }  if(m->client\_key != NULL){  /\*Client key\*/  m->cfg.client\_key.buf = utils\_file\_get(m->client\_key, &m->cfg.client\_key.len);  if(m->cfg.client\_key.buf == NULL && bargs->transport\_mode == TLS\_CLIENT\_VERFIY){  os\_printf("Provide a valid path for Client key\r\n");  return -2;  }  }else if (bargs->transport\_mode == TLS\_CLIENT\_VERFIY){  os\_printf("Provide a valid path for Client key\r\n");  return -2;  }  return 0; |

app\_auto\_generate\_params() calls the app\_fetch\_t2\_macid() function to fetch Talaria TWO’s MAC ID and generate the MQTT client ID, publish and subscribe topics based on the generated client ID.

|  |
| --- |
| uint8\_t t2\_mac\_id[LEN\_OF\_MAC\_ID];  char buf[6];  int index = 0;    /\*Get MAC ID\*/  app\_fetch\_t2\_macid(t2\_mac\_id);  for (int i=0; i < LEN\_OF\_MAC\_ID; i++){  index += snprintf(&buf[index], 128-index, "%x", t2\_mac\_id[i]);  }    snprintf(m->client\_id, MAX\_MQTT\_CLIENT\_ID\_LEN, "T2\_%s\_%d",buf, conn\_num);  os\_printf("\n%s:%d, size = %d", \_\_FUNCTION\_\_, \_\_LINE\_\_, sizeof(m->publish\_topic));  snprintf(m->publish\_topic, MAX\_TOPIC\_LEN, "%s%s\_%d",  m->client\_id,"/pt2", conn\_num);  snprintf(m->subscribe\_topic, MAX\_TOPIC\_LEN, "%s%s\_%d",  m->client\_id,"/st2", conn\_num);  m->mqtt\_lwt\_enable = 1;  m->lwt\_qos = QOS1;  m->lwt\_retain\_enable = 0;  if(conn\_num == 1){  m->lwt\_msg\_len = strlen(APP\_LWT\_MESSAGE\_1);  strncpy(m->lwt\_topic\_name, MQTT\_LWT\_TOPIC\_1, MAX\_LWT\_TOPIC\_LEN);  memcpy((uint8\_t\*)m->lwt\_message,(uint8\_t\*)APP\_LWT\_MESSAGE\_1, m->lwt\_msg\_len);  }else{  m->lwt\_msg\_len = strlen(APP\_LWT\_MESSAGE\_2);  strncpy(m->lwt\_topic\_name, MQTT\_LWT\_TOPIC\_2, MAX\_LWT\_TOPIC\_LEN);  memcpy((uint8\_t\*)m->lwt\_message,(uint8\_t\*)APP\_LWT\_MESSAGE\_2, m->lwt\_msg\_len);  }  os\_printf("\r\n--------------------------------------------------\r\n");  os\_printf("T2 MQTT Client id : %s\r\n",m->client\_id);  os\_printf("T2 MQTT publish topic : %s\r\n", m->publish\_topic);  os\_printf("T2 MQTT subscribe topic: %s\r\n", m->subscribe\_topic);  os\_printf("T2 LWT topic : %s\r\n", m->lwt\_topic\_name);  os\_printf("----------------------------------------------------\r\n"); |

The generated MQTT client ID, Talaria TWO publish topic and the subscribe topic information is printed on to the console. Users can publish and subscribe to these topics from the other MQTT clients.

app\_fetch\_t2\_macid()fetches Talaria TWO’s MAC ID by calling wcm\_get\_hwaddr() API. Following is the definition of app\_fetch\_t2\_macid():

|  |
| --- |
| const uint8\_t \*mac\_addr = wcm\_get\_hwaddr(h);  os\_printf("mac id:");  for(int index = 0;index < 6;index++){  mac\_id[index] = \*(mac\_addr+index);  os\_printf("%x",mac\_id[index]);  } |

app\_mqtt\_connect()is called and the address of the structure variable of type app\_mqtt\_conn\_t and struct app\_mqtt\_param\_t are passed.

app\_mqtt\_connect() initializes the MQTT connection based on the transport mode configured through the boot argument. The MQTT application allows the user to configure the connection based on the mqtt\_tranport\_mode enum.

The app\_mqtt\_connect()allocates the required buffers using app\_mqtt\_conn\_init () and depending on the transport parameter value selected, the non-secure/secure MQTT network initialization API - MQTTNetworkInit()/MQTTNetworkInit\_Tls() is called followed by MQTTNetworkConnect/MQTTNetworkConnect\_Tls().

|  |
| --- |
| os\_printf("\n%s %d", \_\_FUNCTION\_\_, \_\_LINE\_\_);  if(app\_mqtt\_conn\_init(cn) < 0){  goto exit;  }  /\*MQTT N/w connect, based on the transport\*/  if(m->transport == APP\_MQTT\_TM\_TCP){  /\*Non secured MQTT\*/  MQTTNetworkInit(cn->mqtt\_network);  ret = MQTTNetworkConnect(cn->mqtt\_network, (char \*)m->cloud\_url,  m->cloud\_port);  if (ret != 0) {  os\_printf("NetworkConnect = %d\n", ret);  goto exit;} |

The MQTT websocket initialization API M MQTTNetworkConnect\_Ws() is called followed by MQTTNetworkInit\_Ws() to initialize the non-secure/secure websocket client connection. websock\_config\_t contains all the parameters needed by the websocket.

|  |
| --- |
| else if(m->transport == APP\_MQTT\_TM\_WEBSOCK ||  m->transport == APP\_MQTT\_TM\_SECURED\_WEBSOCK ||  m->transport == APP\_MQTT\_TM\_SECURED\_WEBSOCK\_NO\_CERT\_VERIFY) {  websock\_config\_t ws\_cfg;  /\*Init mqtt\*/  MQTTNetworkInit\_Ws(cn->mqtt\_network);  /\*Connect to broker over websocket\*/  memset(&ws\_cfg, 0, sizeof(ws\_cfg));  os\_printf("\nmqttbroker\_address = %s", m->cloud\_url);  ws\_cfg.hostname = (char \*)m->cloud\_url;  ws\_cfg.uri = (char \*)m->websock\_url;  ws\_cfg.port = m->cloud\_port;  ws\_cfg.time\_out = 300;  ws\_cfg.secured = (m->transport == APP\_MQTT\_TM\_WEBSOCK) ? 0 : 1;  memcpy((char \*)&ws\_cfg.ssl\_config,(const char \*) &m->cfg, sizeof((m->cfg)));  ret = MQTTNetworkConnect\_Ws(cn->mqtt\_network, &ws\_cfg);  if(ret < 0) {  os\_printf("\r\nmqtt\_connect\_ws %d ",  ret);  goto exit;  }  } |

Then, MQTTClientInit() is called to configure the client connection with the parameters like time out and the required buffers. MQTT client is also initiated with the LWT parameters. When the client performs rough-hewn disconnect, Talaria TWO LWT topic will be published with the LWT message. MQTTConnect() is called by passing the pointer to the client configuration and the MQTT packet connect data to connect to a broker.

|  |
| --- |
| MQTTClientInit(cn->mqtt\_client, cn->mqtt\_network, 15\*1000, cn->sendbuf,  APP\_MQTT\_SEND\_BUF\_SIZE, cn->readbuf,  APP\_MQTT\_READ\_BUF\_SIZE);  MQTTPacket\_connectData data = MQTTPacket\_connectData\_initializer;  data.willFlag = 0;  data.willFlag = m->mqtt\_lwt\_enable;  if(data.willFlag) {  data.will.qos = m->lwt\_qos;  data.will.struct\_version = 3;  data.will.retained = m->lwt\_retain\_enable;  data.will.topicName.lenstring.len = strlen(m->lwt\_topic\_name);  data.will.topicName.lenstring.data = m->lwt\_topic\_name;  data.will.message.lenstring.len = m->lwt\_msg\_len;  data.will.message.lenstring.data = m->lwt\_message;  } |

data.willFlag is enabled when mqtt\_lwt\_enable is enabled. LWT client is configured with QOS1.

|  |
| --- |
| data.MQTTVersion = 3;  data.clientID.cstring = (char \*)m->client\_id;  data.username.cstring = (char \*)m->cloud\_usr\_name;  data.password.cstring = (char \*)m->cloud\_usr\_psw;  data.keepAliveInterval = APP\_MQTT\_DEFAULT\_KA\_INTR;  data.cleansession = APP\_MQTT\_DEFAULT\_CLEAN\_SESSION;  data.kaRespTimeout = 0;  os\_printf("\r\nConnecting ...\n");  ret = MQTTConnect(cn->mqtt\_client, &data);  if(0 == ret){  cn->connected = 1;  os\_printf("\nMQTTConnect Success. ret = %d", ret);  }else{  os\_printf("\nMQTTConnect Failed. ret = %d", ret);  } |

On successfully establishing the MQTT connection, app\_thread\_entry\_fn thread subscribes to the unique topics generated, by calling app\_subscribe(&c1, &mqtt\_param\_1). Since there are two connections, app\_subscribe(&c1, &mqtt\_param\_2) is called again with a different subscription topic.

|  |
| --- |
| /\* MQTT connect\*/  if(!c1.connected){  app\_mqtt\_connect(&c1, &mqtt\_param\_1);  /\* MQTT Subscribe\*/  app\_subscribe(&c1, &mqtt\_param\_1);  }  /\* MQTT connect - a second connection\*/  if(bargs.num\_conn == 2 && !c2.connected){  app\_mqtt\_connect(&c2, &mqtt\_param\_2);  /\* MQTT Subscribe\*/  app\_subscribe(&c2, &mqtt\_param\_2);  } |

app\_subscribe() function calls MQTTSubscribe() API to subscribe to the given topic and register a callback.

|  |
| --- |
| MQTTSubscribe(cn->mqtt\_client, m->subscribe\_topic,  m->qos, app\_mqtt\_subscribe\_cb);  return 0; |

Here, the callback function, app\_mqtt\_subscribe\_cb gets invoked when a message is received from the broker of the subscribed topic. The callback extracts the topic name, topic length and the message payload.

|  |
| --- |
| void app\_mqtt\_subscribe\_cb(MessageData\* Msg)  {  int i;    os\_printf("\nMQTTSubscribe Call back");  if(Msg->topicName->cstring){  os\_printf("\n\ttopic = %s", Msg->topicName->cstring);  }else{  os\_printf("\n\ttopic = ");  for(i= 0; i < Msg->topicName->lenstring.len; i++)  os\_printf("%c", Msg->topicName->lenstring.data[i]);  os\_printf("\n");  }  os\_printf("\n\tMessage = ");  char \*p= Msg->message->payload;  for(i= 0; i < Msg->message->payloadlen; i++)  os\_printf("%c", p[i]);  os\_printf("\n");  } |

Now, app\_thread\_entry\_fn thread publishes the messages to both the connections by calling app\_mqtt\_publish() function. The thread publishes data every 1 second.

|  |
| --- |
| if(c1.connected){  app\_mqtt\_publish(&c1, &mqtt\_param\_1, "Hello From T2");  }  if(c2.connected){  os\_printf("\n%s:%d", \_\_FUNCTION\_\_, \_\_LINE\_\_);  app\_mqtt\_publish(&c2, &mqtt\_param\_2, "Hello From T2");  }  os\_msleep(1000); |

app\_mqtt\_init() API Initializes MQTT. The client connects to the broker with the specific protocol based on transport mode specified in boot argument (tcp/tls).

|  |
| --- |
| if(app\_mqtt\_parse\_params(param) == 0){  if(param.cloud\_url == NULL || param.cloud\_port == 0 ){  return -1;  }  os\_printf("\nparam parsing done...");    /\* Initialize the mqtt  \*/  if(app\_mqtt\_init() < 0){  return -1;  }  mqtt\_network = os\_alloc(sizeof(MQTTNetwork));  if (NULL == mqtt\_network) {  os\_printf("\nMalloc Fail @%s:%d", \_\_FUNCTION\_\_,  \_\_LINE\_\_);  return -1;  }  mqtt\_client = os\_alloc(sizeof(MQTTClient));  if(NULL == mqtt\_client){  os\_printf("\nMalloc Fail @%s:%d", \_\_FUNCTION\_\_,  \_\_LINE\_\_);  return -1;  }    if(param.transport == APP\_MQTT\_TRANSPORT\_TCP){  /\*Non secured MQTT\*/  MQTTNetworkInit(mqtt\_network);  ret = MQTTNetworkConnect(mqtt\_network, (char \*)param.cloud\_url, param.cloud\_port);  if (ret != 0) {  os\_printf("NetworkConnect = %d\n", ret);  return -3;  }  }else if((param.transport == APP\_MQTT\_TRANSPORT\_TLS) || (param.transport == APP\_MQTT\_TRANSPORT\_TLS\_NO\_CERT\_VERIFY)){  /\*Secured MQTT\*/  MQTTNetworkInit\_Tls(mqtt\_network);  ret = MQTTNetworkConnect\_Tls(mqtt\_network, (char\*)param.cloud\_url,param.cloud\_port,&cfg);  if (ret < 0) {  os\_printf("\r\nmqtt\_connect\_tls %d !!", ret);  return -3;  }  }else{  os\_printf("Set proper MQTT Transport mode\r\n");  }} |

Mqttclientinit() API initializes the MQTT Client. This API is called with the MQTT network object mqtt\_network, pointers to read, send buffers and the MQTTClient handle mqtt\_client as arguments.

|  |
| --- |
| MQTTClientInit(mqtt\_client, mqtt\_network, 15\*1000, sendbuf, STW\_MQTT\_SEND\_BUF\_SIZE, readbuf, STW\_MQTT\_READ\_BUF\_SIZE);  MQTTPacket\_connectData data = MQTTPacket\_connectData\_initializer;  data.willFlag = 0;  data.MQTTVersion = 3;  data.clientID.cstring = (char \*)mqtt\_param.client\_id;  data.username.cstring = (char \*)mqtt\_param.cloud\_usr\_name;  data.password.cstring = (char \*)mqtt\_param.cloud\_usr\_psw;  data.keepAliveInterval = STW\_MQTT\_DEFAULT\_KA\_INTR;  data.kaRespTimeout = 20;  data.cleansession = STW\_MQTT\_DEFAULT\_CLEAN\_SESSION;  os\_printf("\r\nConnecting ...\n");  ret = MQTTConnect(mqtt\_client, &data);  os\_printf("\nMQTTConnect ret = %d", ret);  return ret; |

[data.kaRespTimeout = 20] keepalive ensures the connection between the broker and client is still open and that the broker and the client are aware of being connected.

## Publishing Data to the MQTT Instance

Function app\_mqtt\_publish\_message() takes a pointer to the message, a pointer to the topic, the length of a message as arguments and publishes it to the remote MQTT Broker running a MQTT instance.

The message is published under the topic T2\_<mac id of T2>/publisher. MQTTPublish()API is used to publish a message. The MQTT client ID, publish topic and subscribe topic are unique to the Talaria TWO module.

For example, MQTT client ID computed for Talaria TWO module is T2\_e0693a02dfa. Hence, Talaria TWO’s publish topic is T2\_e0693a02dfa/subscribe and the subscribe topic is T2\_e0693a02dfa/publisher. These details are displayed on the Download Tool’s console.

Commands to publish and subscribe to the given topic are as follows:

Subscribing to a topic from a host(PC):

|  |
| --- |
| mosquitto\_sub.exe -h test.mosquitto.org -p 1883 -u <user name> -P <Password> -t T2\_<mac id of T2>/subscribe |

Publishing data to the given topic :

|  |
| --- |
| mosquitto\_pub.exe -h test.mosquitto.org -p 1883 -u <user name> -P <Password> -t T2\_<mac id of T2>/publisher -m “Message to Publish” |

**Note**: The actual MQTT Client ID, publish and subscribe topics are computed by the application and displayed on the Download Tool’s console.

app\_mqtt\_publish\_message() publishes the MQTT messages to the server. The pmessage contains the address of the buffer that contains the message to be published. The length variable contains the length of the publish message.

|  |
| --- |
| char device\_data\_recieved[MAX\_PUBLISH\_MSG\_SIZE];  if(len < MAX\_PUBLISH\_MSG\_SIZE){  /\* Message is published under the topic innophase\_t2/temperature.  MQTTPublish()is used for this. \*/  MQTTMessage \*publish = os\_zalloc(sizeof(MQTTMessage));  publish->payload = pmessage;  publish->payloadlen = len;  /\* As we are restarting provisioning, reset the housekeeping  and status mssg to default value 'waiting'. \*/  memcpy(device\_data\_recieved, pmessage, len);  device\_data\_recieved[len]='\0';  rc = MQTTPublish(mqtt\_client, pub\_topic, publish);  if(rc != 0)  {  os\_printf("\nMQTTPublish failed. Ret= %d", rc);  }  else  {  os\_printf("\n%u:Message published successfully [%s]",os\_systime(), device\_data\_recieved);  }  os\_free(publish);  }else{  os\_printf("\n Could not publish the message. Please send a message less than 248 bytes");  } |

## Subscribing to MQTT Topic

Function app\_subscribe() subscribes to a topic and registers the call back function app\_mqtt\_subscribe\_cb(MessageData\* Msg). The call back gets invoked when there is a message published by a client on the same topic.

|  |
| --- |
| os\_printf("\n%s:%d", \_\_FUNCTION\_\_, \_\_LINE\_\_);  MQTTSubscribe(cn->mqtt\_client, m->subscribe\_topic,  m->sub\_qos, app\_mqtt\_subscribe\_cb);    os\_printf("\n%s: %d", \_\_FUNCTION\_\_, \_\_LINE\_\_);  return 0; |

In the main function, app\_subscribe() is called once to register a handler for MQTT subscribe and app\_mqtt\_publish() is called every two seconds to publish a message.

|  |
| --- |
| int i;  os\_printf("\nMQTTSubscribe Call back");  if(Msg->topicName->cstring){  os\_printf("\n\ttopic = %s", Msg->topicName->cstring);  }else{  os\_printf("\n\ttopic = ");  for(i= 0; i < Msg->topicName->lenstring.len; i++)  os\_printf("%c", Msg->topicName->lenstring.data[i]);  os\_printf("\n");  }  os\_printf("\n\tMessage = ");  char \*p= Msg->message->payload;  for(i= 0; i < Msg->message->payloadlen; i++)  os\_printf("%c", p[i]);  os\_printf("\n"); |

## Last Will & Testament

In MQTT, Last Will and Testament (LWT) is used to notify other clients about a rough-hewn disconnected client. All the clients can specify their last-will message when it connects to a broker. The last-will message is a normal MQTT message with a topic, retained message flag, QoS, and payload.

The broker stores the message until it detects that the client has disconnected ungracefully. In response to the rough-hewn disconnect, the broker sends the last-will message to all subscribed clients of the last-will message topic.

If the client disconnects gracefully with a correct DISCONNECT message, the broker discards the stored LWT message.

LWT publish topics and messages are defined in the application.

|  |
| --- |
| /\*last will topics for connections\*/  #define MQTT\_LWT\_TOPIC\_1 "will\_con1"  #define MQTT\_LWT\_TOPIC\_2 "will\_con2"  /\*LWT messages\*/  #define APP\_LWT\_MESSAGE\_1 "Connection-1 Terminated"  #define APP\_LWT\_MESSAGE\_2 "Connection-1 Terminated" |

If Talaria TWO client disconnects with a rough-hewn disconnect, the server publishes LWT messages defined in the application to all the LWT topics after 90 seconds, as Talaria TWO has a default keepalive of 60 seconds. Hence, the server will wait for a period of 90 seconds to ensure the connection is lost.

# Running the Application using Mosquitto Project’s Test Server

Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1.

The Mosquitto project allows to test the MQTT based applications to test using its test server. Users can use a custom server or any of the following tested public MQTT brokers:

1. **mqtt.eclipseprojects.io**
   1. 1883 : MQTT over unencrypted TCP
   2. 8883 : MQTT over encrypted TCP
   3. 80 : MQTT over unencrypted Websocket (note: URL must be */mqtt* )
   4. 443: MQTT over encrypted WebSockets (note: URL must be */mqtt* )
2. **mqtt-dashboard.com**
   1. TCP Port: 1883
   2. TLS TCP Port: 8883
   3. Websocket Port: 8000
   4. TLS Websocket Port: 8884
3. test.mosquitto.org
   1. 1883: MQTT, unencrypted, unauthenticated
   2. 1884: MQTT, unencrypted, authenticated
   3. 8883: MQTT, encrypted, unauthenticated
   4. 8884: MQTT, encrypted, client certificate required
   5. 8080: MQTT over WebSockets, unencrypted, unauthenticated
   6. 8081: MQTT over WebSockets, encrypted, unauthenticated
   7. 8091: MQTT over WebSockets, unencrypted, authenticated

**Note**: test.mosquitto.org is used in this document for illustration purposes only.

The following steps describe the procedure to test the MQTT application using the Mosquitto project’s test server.

To evaluate the secure MQTT, the certificates (CA certificate, Client certificate, Client key) bundled along with the MQTT sample app can be used. The TLS certificates are generated from the following URL: <https://test.mosquitto.org/ssl/>.

**Note**:

1. The certificates provided as a part of the MQTT example application are generated from <https://test.mosquitto.org/> and <https://test.mosquitto.org/ssl/>.
2. The CA certificate: mosquitto.org.crt can be downloaded from <https://test.mosquitto.org/>.
3. The client certificate: client.crt and the client key: client.key can be generated from <https://test.mosquitto.org/ssl/> by following the instruction mentioned in the same website.

The validity of the certificates generated is only 90 days, hence it is recommended to generate the three certificates while evaluating the MQTT sample application, to ensure that the expired certificates are not used.

Refer to section 0 to access the validity of the certificates.

## Installing and Running the Mosquitto MQTT Tool

1. Download [mosquitto-2.0.11-install-windows-x64.exe](file:///C:\C:\C:\C:\Users\innop\mqtt\embedded_apps\C:\C:\Users\91963\Downloads\mosquitto-2.0.11-install-windows-x64.exe) from <https://mosquitto.org/download/> and install the same.
2. Open a command prompt window on the PC and subscribe to a topic by issuing the following command:

|  |
| --- |
| mosquitto\_sub.exe -h test.mosquitto.org -p 1883 -u <user name> -P <Password> -t T2\_<mac id of T2>/subscribe |

Ensure the note in section 0 is followed and the binary is generated.

In the example, the username and password used are innophase. The topic programmed in the application binary mqtt.elf is T2\_<mac\_id of T2>/subscribe.

The following picture shows the command prompt window:

For example, subscribe to a topic of one of the two connections initiated over non-secure port:



Figure 1: Non-secure - Command prompt window

## Programming the Talaria TWO module

1. Certificates need to be flashed for secure MQTT. In case of non-secure MQTT, the user can begin the procedure from step 2.b. The certificates for the sample app are present in: *freertos\_sdk\_x.y/examples/mqtt/certs/data* directory.

**Note**: x and y refers to the SDK version. For example: freertos\_sdk\_2.5.

Provide the path of the certificates stored on host PC in the Write files from a directory column to store the certificates on Talaria TWO’s file system i.e., /data/

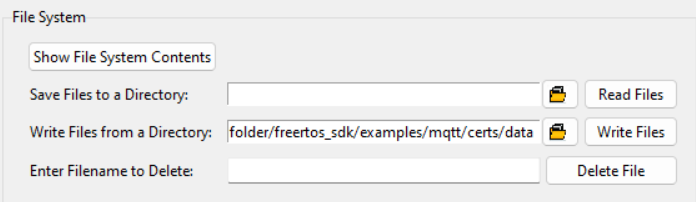


Figure 2: Flashing the certificates to Talaria TWO file system by loading from a directory

After writing the certificates to Talaria TWO’s file system, verify the path of the certificates by clicking on Show File System Contents. A dialog box pops up and displays the path of the certificates written along with the size of each of these certificates.

Graphical user interface, text

Description automatically generated

Figure 3: Certificates along with size displayed

1. Program mqtt.elf (*freertos\_sdk\_x.y/examples/mqtt/bin*) using the Download Tool (*freertos\_sdk\_x.y/pc\_tools/Download\_Tool/bin)*.
   1. Launch the Download Tool provided with InnoPhase Talaria TWO SDK.
   2. In the GUI window:
      1. Boot Target: Select the appropriate EVK from the drop-down.
      2. ELF Input: Load the mqtt.elf by clicking on Select ELF File.
      3. AP Options: Provide the SSID and Passphrase under AP Options to connect to an Access Point.
      4. Boot arguments: Pass the following boot arguments:
         1. Non-secured MQTT:

|  |
| --- |
| cloud\_url=test.mosquitto.org,cloud\_port=1883,cloud\_usr\_name=<user name >,cloud\_usr\_psw=<password>,mqtt\_no\_poll=1, num\_conn=1 |

* + - 1. For secured MQTT (Verifying server certificate) :

|  |
| --- |
| cloud\_url=test.mosquitto.org,cloud\_port=8883, cloud\_usr\_name=<username>,cloud\_usr\_psw=<password>,transport\_mode=1,pub\_qos=1,sub\_qos=1,ca\_cert=/data/mosquitto.org.crt,mqtt\_no\_poll=1,num\_conn=1 |

* + - 1. For secured MQTT (No certificate verify) :

|  |
| --- |
| cloud\_url=test.mosquitto.org,cloud\_port=8883, cloud\_usr\_name=<username>,cloud\_usr\_psw=<password>,transport\_mode=3,pub\_qos=1,sub\_qos=1,mqtt\_no\_poll=1, num\_conn=1 |

* + - 1. For secured MQTT (verify the server certificate and provide the client certificate) :

|  |
| --- |
| cloud\_url=test.mosquitto.org,cloud\_port=8884, cloud\_usr\_name=<username>,cloud\_usr\_psw=<password>,,transport\_mode=2,pub\_qos=1,sub\_qos=1,ca\_cert=/data/mosquitto.org.crt,client\_cert=/data/client.crt,client\_key=/data/client.key,mqtt\_no\_poll=1,num\_conn=1 |

* + - 1. For MQTT over WebSockets, unencrypted, unauthenticated:

|  |
| --- |
| cloud\_url=test.mosquitto.org,cloud\_port=8080,cloud\_usr\_name=<username>,cloud\_usr\_psw=<password>, mqtt\_no\_poll=1,num\_conn=1,transport\_mode=4, websock\_url=ws://test.mosquitto.org/mqtt |

* + - 1. For MQTT over WebSockets, encrypted, with certificate verification:

|  |
| --- |
| cloud\_url=test.mosquitto.org ,cloud\_port=8081,cloud\_usr\_name=innophase,cloud\_usr\_psw=innophase,mqtt\_no\_poll=1,num\_conn=1,transport\_mode=5,pub\_qos=1,sub\_qos=1, websock\_url=ws://test.mosquitto.org/mqtt,ca\_cert=/data/mosquitto.org.crt,client\_cert=/data/client.crt,client\_key=/data/client.key |

* + - 1. For MQTT over WebSockets, encrypted, without certificate verification:

|  |
| --- |
| cloud\_url=test.mosquitto.org ,cloud\_port=8091,cloud\_usr\_name=<username>,cloud\_usr\_psw=<password>,mqtt\_no\_poll=1,num\_conn=1,transport\_mode=6,pub\_qos=1,sub\_qos=1,websock\_url=ws://test.mosquitto.org/mqtt |

* + 1. Programming: Click on Prog Flash.

1. The console window displays MQTTConnect ret = 0 indicating that Talaria TWO can connect to the test.mosquitto.org server.

Non-secured MQTT:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWWAE  Build $Id: git-df9b9ef $  Flash detected. flash.hw.uuid: 39483937-3207-00b0-0064-ffffffffffff  Bootargs: cloud\_url=test.mosquitto.org cloud\_port=1883 mqtt\_no\_poll=1 num\_conn=1 np\_conf\_path=/data/nprofile.json ssid=Lucy passphrase=Password@321  $App:git-94e4627  SDK Ver: FREERTOS\_SDK\_1.0  MQTT Example App  addr e0:69:3a:00:16:d4  Connecting to added network : Lucy  [0.580,194] CONNECT:c8:e7:d8:8c:ba:3c Channel:6 rssi:-82 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  app\_wifi\_status\_cb: status = 124457wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  app\_wifi\_status\_cb: status = 6881281[0.695,319] MYIP 192.168.1.103  [0.695,484] IPv6 [fe80::e269:3aff:fe00:16d4]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  app\_wifi\_status\_cb: status = 105  Connected to added network : Lucy  MQTTNoPollInit:774  MQTTRun\_NoPollThread: 718mac id:e0693a016d4  ------------------------------------------------------  T2 MQTT Client id : T2\_e0693a016d4\_1  T2 MQTT publish topic : T2\_e0693a016d4\_1/pt2\_1  T2 MQTT subscribe topic: T2\_e0693a016d4\_1/st2\_1  T2 LWT topic : will\_con1  --------------------------------------------------------  app\_mqtt\_connect 155  app\_mqtt\_conn\_init  /home/synergic/Workspace/FreeRTOS/freertos\_embedded\_apps/components/mqtt/platform/mqtt\_nw\_tcp.c:MQTTNetworkConnect  Connecting ...  \_mqtt\_cycle : packet\_type = 2  MQTTConnect Success. ret = 0  app\_subscribe:307  \_mqtt\_cycle : packet\_type = 9  app\_subscribe: 311  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 1, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 2, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 3, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 4, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 5, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 6, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 7, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 8, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 9, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 10, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 11, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 12, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 13, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 14, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 15, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 16, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 17, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 18, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 19, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 20, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 21, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 22, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 23, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 24, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 25, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 26, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully |

Secured MQTT:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWWAE  Build $Id: git-df9b9ef $  Flash detected. flash.hw.uuid: 39483937-3207-00b0-0064-ffffffffffff  Bootargs: cloud\_url=test.mosquitto.org cloud\_port=8883 transport\_mode=1 pub\_qos=1 sub\_qos=1 ca\_cert=/data/mosquitto.org.crt mqtt\_no\_poll=1 num\_conn=1 np\_conf\_path=/data/nprofile.json ssid=Lucy passphrase=Password@321  $App:git-94e4627  SDK Ver: FREERTOS\_SDK\_1.0  MQTT Example App  addr e0:69:3a:00:16:d4  Connecting to added network : Lucy  [0.708,336] CONNECT:c8:e7:d8:8c:ba:3c Channel:6 rssi:-80 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  app\_wifi\_status\_cb: status = 124457wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  app\_wifi\_status\_cb: status = 6881281[0.907,204] MYIP 192.168.1.103  [0.907,369] IPv6 [fe80::e269:3aff:fe00:16d4]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  app\_wifi\_status\_cb: status = 105  Connected to added network : Lucy  MQTTNoPollInit:774  MQTTRun\_NoPollThread: 718  ------------------------------------------------------  T2 MQTT Client id : T2\_e0693a0a66\_1  T2 MQTT publish topic : T2\_e0693a0a66\_1/pt2\_1  T2 MQTT subscribe topic: T2\_e0693a0a66\_1/st2\_1  T2 LWT topic : will\_con1  --------------------------------------------------------  mac id:e0693a0a66  app\_auto\_generate\_params:111, size = 64  ------------------------------------------------------  T2 MQTT Client id : T2\_e0693a016d4\_1  T2 MQTT publish topic : T2\_e0693a016d4\_1/pt2\_1  T2 MQTT subscribe topic: T2\_e0693a016d4\_1/st2\_1  T2 LWT topic : will\_con1  --------------------------------------------------------  app\_mqtt\_connect 155  app\_mqtt\_conn\_init  /home/synergic/Workspace/FreeRTOS/freertos\_embedded\_apps/components/mqtt/platform/mqtt\_nw\_tls.c:MQTTNetworkConnect\_Tls  . [SSL\_WRAP]Checking input configurations...  . [SSL\_WRAP]Seeding the random number generator...  . [SSL\_WRAP]Loading the CA root certificate ...Cert Len = 1477  . [SSL\_WRAP]Connecting to tcp test.mosquitto.org:8883...  . [SSL\_WRAP]Setting up the SSL/TLS structure...  . [SSL\_WRAP]setting configurations..  >auth mode = 2 (0- skip, 1- optional, 2- required  >max fragment len = 0  >Handshake timeout = 30 Sec  . [SSL\_WRAP]Performing the SSL/TLS handshake...  . [SSL\_WRAP] Handshake done. ok  . [SSL\_WRAP]Verifying peer X.509 certificate.  Connecting ...  \_mqtt\_cycle : packet\_type = 2  MQTTConnect Success. ret = 0  app\_subscribe:264  \_mqtt\_cycle : packet\_type = 9  \_mqtt\_cycle : packet\_type = 5  \_mqtt\_cycle : packet\_type = 7  Message Published Successfully  Publish stats: Success = 1, Failure = 0  mqtt\_ssl\_sock\_read: setting rval to 0  \_mqtt\_cycle : packet\_type = 5  \_mqtt\_cycle : packet\_type = 7  Message Published Successfully  Publish stats: Success = 2, Failure = 0  mqtt\_ssl\_sock\_read: setting rval to 0  \_mqtt\_cycle : packet\_type = 5  \_mqtt\_cycle : packet\_type = 7  Message Published Successfully  Publish stats: Success = 3, Failure = 0  mqtt\_ssl\_sock\_read: setting rval to 0  \_mqtt\_cycle : packet\_type = 5  \_mqtt\_cycle : packet\_type = 7  Message Published Successfully  Publish stats: Success = 4, Failure = 0  mqtt\_ssl\_sock\_read: setting rval to 0  \_mqtt\_cycle : packet\_type = 5  \_mqtt\_cycle : packet\_type = 7  Message Published Successfully  …  … |

1. For testing the MQTT over WebSocket protocol, Mosquitto Websocket server is used.

To connect MQTT over Websocket, add the following boot argument to the test.mosquitto.org websocket server.

|  |
| --- |
| websock\_url=ws://test.mosquitto.org/mqtt |

Non-secured MQTT over WebSocket:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWWAE  Build $Id: git-df9b9ef $  Flash detected. flash.hw.uuid: 39483937-3207-00b0-0064-ffffffffffff  Bootargs: cloud\_url=test.mosquitto.org cloud\_port=8080 cloud\_usr\_name=innophase ,cloud\_usr\_psw=innophase mqtt\_no\_poll=1 num\_conn=1 transport\_mode=4 websock\_url=ws://test.mosquitto.org/mqtt np\_conf\_path=/data/nprofile.json ssid=Lucy passphrase=Password@321  $App:git-94e4627  SDK Ver: FREERTOS\_SDK\_1.0  MQTT Example App  addr e0:69:3a:00:0a:66  Connecting to added network : Lucy  [0.812,590] CONNECT:96:6a:1b:0d:62:e4 Channel:1 rssi:-25 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  app\_wifi\_status\_cb: status = 200wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  app\_wifi\_status\_cb: status = 202[0.865,350] MYIP 192.168.58.243  [0.865,630] IPv6 [fe80::e269:3aff:fe00:a66]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  app\_wifi\_status\_cb: status = 204  Connected to added network : Lucy  MQTTNoPollInit:714mac id:e0693a0a66  app\_auto\_generate\_params:154, size = 64  ------------------------------------------------------  T2 MQTT Client id : T2\_e0693a0a66\_1  T2 MQTT publish topic : T2\_e0693a0a66\_1/pt2\_1  T2 MQTT subscribe topic: T2\_e0693a0a66\_1/st2\_1  T2 LWT topic : will\_con1  --------------------------------------------------------  app\_mqtt\_connect 145  app\_mqtt\_conn\_init  mqttbroker\_address = test.mosquitto.org  MQTTNetworkConnect\_Ws  MQTTRun\_NoPollThread: 658  Connecting ...  \_mqtt\_cycle : packet\_type = 2  MQTTConnect Success. ret = 0  app\_subscribe:293  \_mqtt\_cycle : packet\_type = 9  app\_subscribe: 297  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 1, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 2, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 3, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 4, Failure = 0 |

Secured MQTT over Websocket:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWWAE  Build $Id: git-aba7fa2 $  Flash detected. flash.hw.uuid: 39483937-3207-008d-009d-ffffffffffff  Bootargs: cloud\_url=172.16.16.30 cloud\_port=9001 cloud\_usr\_name=innophase cloud\_usr\_psw=innophase mqtt\_no\_poll=1 num\_conn=1 transport\_mode=5 websock\_url=ws://172.16.16.30/mqtt ca\_cert=/data/ca.crt client\_cert=/data/client.crt client\_key=/data/client.key np\_conf\_path=/data/nprofile.json ssid=Xiaomi\_Ax6000\_iop passphrase=InnoQA2023$  $App:git-e719539  SDK Ver: FREERTOS\_SDK\_1.0  MQTT Example App  addr e0:69:3a:00:16:06  Connecting to added network : Xiaomi\_Ax6000\_iop  [1.197,870] CONNECT:d4:da:21:54:d3:c6 Channel:1 rssi:-21 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  app\_wifi\_status\_cb: status = 124457wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  app\_wifi\_status\_cb: status = 6881281[1.811,667] MYIP 192.168.31.198  [1.811,833] IPv6 [fe80::e269:3aff:fe00:1606]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  app\_wifi\_status\_cb: status = 105  Connected to added network : Xiaomi\_Ax6000\_iop  MQTTNoPollInit:775  MQTTRun\_NoPollThread: 719mac id:e0693a0166 T2 MQTT Client id : T2\_e0693a0166\_1  T2 MQTT publish topic : T2\_e0693a0166\_1/pt2\_1  T2 MQTT subscribe topic: T2\_e0693a0166\_1/st2\_1  T2 LWT topic : will\_con1 app\_mqtt\_connect 155  app\_mqtt\_conn\_init  mqttbroker\_address = 172.16.16.30  MQTTNetworkConnect\_Ws  . [SSL\_WRAP]Checking input configurations...  . [SSL\_WRAP]Seeding the random number generator...  . [SSL\_WRAP]Loading the CA root certificate ...Cert Len = 1221  . Loading the Client(Own) certificate ...Cert Len = 1108  . [SSL\_WRAP]Loading the Client(Own) Key ...Key Len = 1676  . [SSL\_WRAP]Connecting to tcp 172.16.16.30:9001...  . [SSL\_WRAP]Setting up the SSL/TLS structure...  . [SSL\_WRAP]setting configurations..  >auth mode = 2 (0- skip, 1- optional, 2- required  >max fragment len = 0  >Handshake timeout = 30 Sec  . [SSL\_WRAP]Performing the SSL/TLS handshake...  . [SSL\_WRAP] Handshake done. ok  . [SSL\_WRAP]Verifying peer X.509 certificate.  Connecting ...  \_mqtt\_cycle : packet\_type = 2  MQTTConnect Success. ret = 0  app\_subscribe:307  \_mqtt\_cycle : packet\_type = 9  app\_subscribe: 311  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 1, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 2, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully |

## Publishing the Message

1. Application continues publishing the message “***Hello From T2”*** every second. The published message can be seen on the subscriber’s window (command prompt on the PC).

The MAC ID of the Talaria TWO device used in this example is e0:69:3a:00:0e:ba. The unique MQTT Client ID generated based on this MAC ID is T2\_e0693a0eba\_1 for the first connection and T2\_e0693a0eba\_2 for the second connection .

Talaria TWO publishes the messages to the topics T2\_e0693a0eba\_1/pt2\_1 from the first connection and T2\_e0693a0eba\_2/pt2\_2 from the second connection.

Hence, the other MQTT Client (PC here) will subscribe to the same topic.

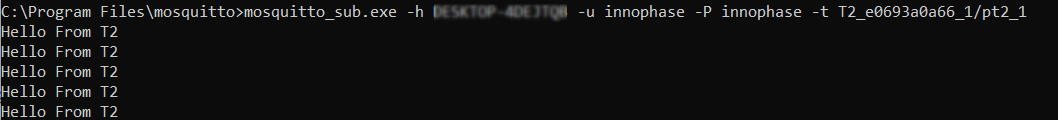


Figure : Talaria TWO’s Published messages

## Subscribing to a Topic

1. Talaria TWO device has subscribed to the topic T2\_<mac id> \_1/st2\_1 (in this example, Talaria TWO’s subscribed topic is based on the MAC ID T2\_e0693a0eba\_1/st2\_1) from the first connection and T2\_<mac id> \_1/st2\_2 from the second connection (i.e., T2\_e0693a0eba\_2/st2\_2). The message published by other MQTT client to this topic will be displayed on Talaria TWO’s console.
2. Publish a message to the topic innophase\_t2/subscriber for another MQTT client (PC) as shown in Figure 5.

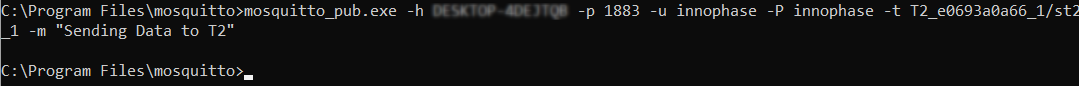


Figure : The published message will be displayed on Talaria TWO’s console

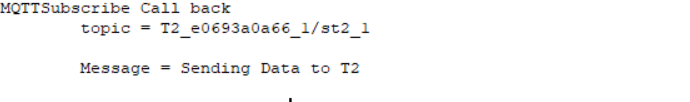


Figure 6: Message received by Talaria TWO for the topic subscribed

## Evaluating LWT Feature

To evaluate the LWT feature of Talaria TWO example application, an MQTT client should be subscribed to the LWT topic of Talaria TWO.

|  |
| --- |
| mosquitto\_sub.exe -h Innophase-SYNEM2043 -p 1883 -u innophase -P innophase -t will\_con1 |

If Talaria TWO client disconnects with a rough-hewn disconnect, the server publishes LWT messages defined in the application to all the LWT topics after 90 seconds.



Figure : MQTT client subscriber for LWT Topic

Server-side console:

Server sends the LWT message to all the clients subscribed to LWT topic.

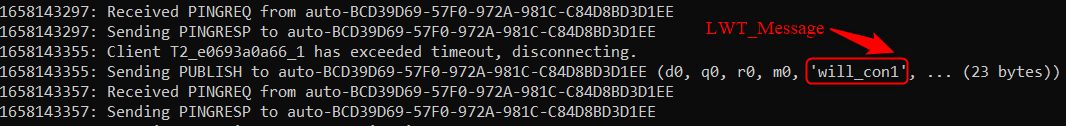


Figure : LWT publish message published to all the clients

## Evaluating Websocket Feature

To evaluate the websocket feature of Talaria TWO, example application MQTT Box extension is used. MQTTBox is a developers helper program to create and test MQTT connectivity protocol, which is available as an google extension.



Figure : MQTTBox Extension

To test the websocket client, launch the MQTT Box Extension and create a new MQTT over websocket client.

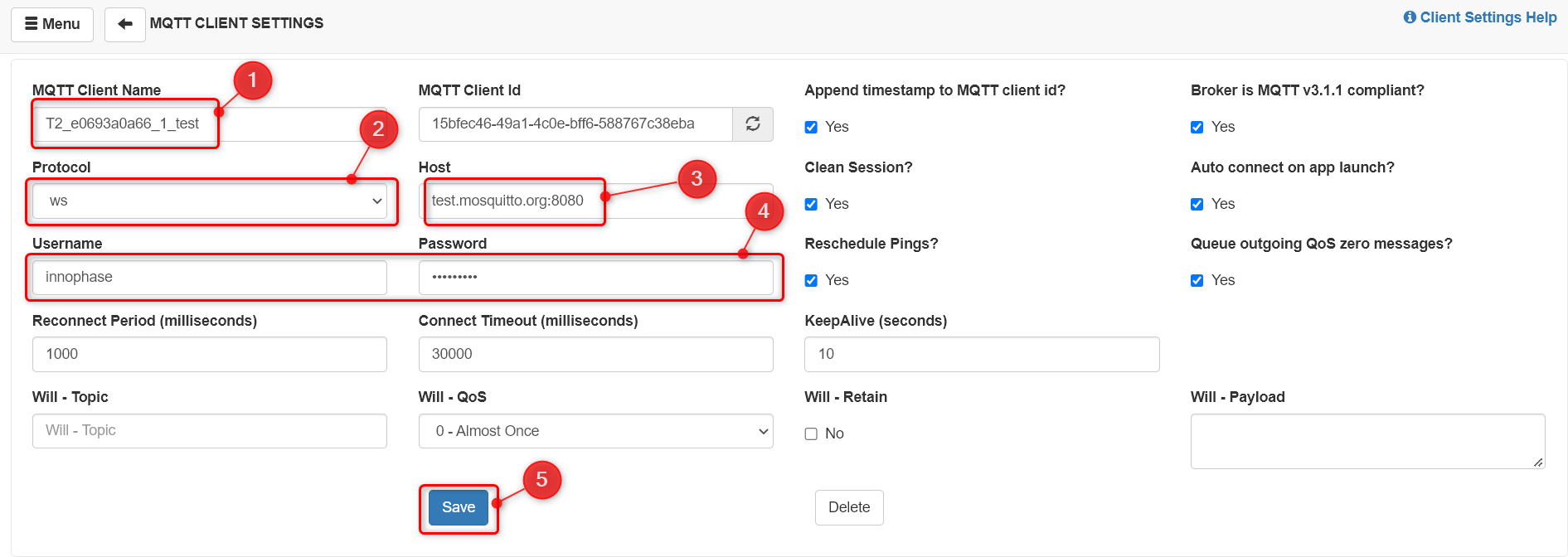


Figure : MQTTBox Extension settings

* + - 1. Add client name.
      2. Select **WS** Protocol.
      3. Add the host address (test.mosquitto.org:8080)
      4. Add username password for the MQTT over websocket client.
      5. Click on “Save” button.

After saving, client will connect to the server if the connection is successful. The connection message will be shown in the publish and subscribe window.

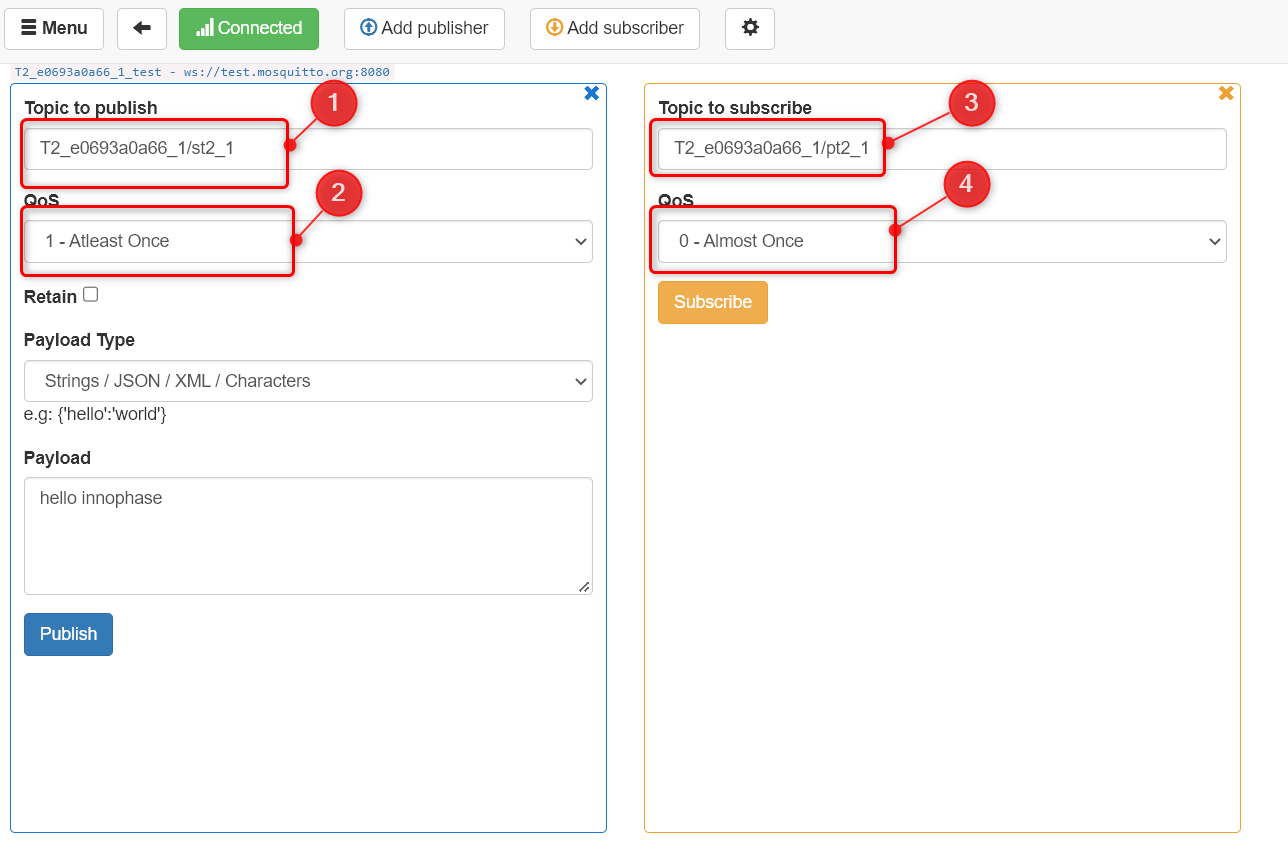


Figure : MQTTBox Extension publish subscribe window

* + - 1. Add Talaria TWO subscribe topic as a publish topic.
      2. Select the Publish QOS value.
      3. Add Talaria TWO publish topic as a subscribe topic.
      4. Select the subscribe QOS value.
      5. For publishing the payload click on “Publish” Button.
      6. For subscribing click on “Subscribe” Button.

Once the connection is successful, Talaria TWO published messages are subscribed at the subscription window.

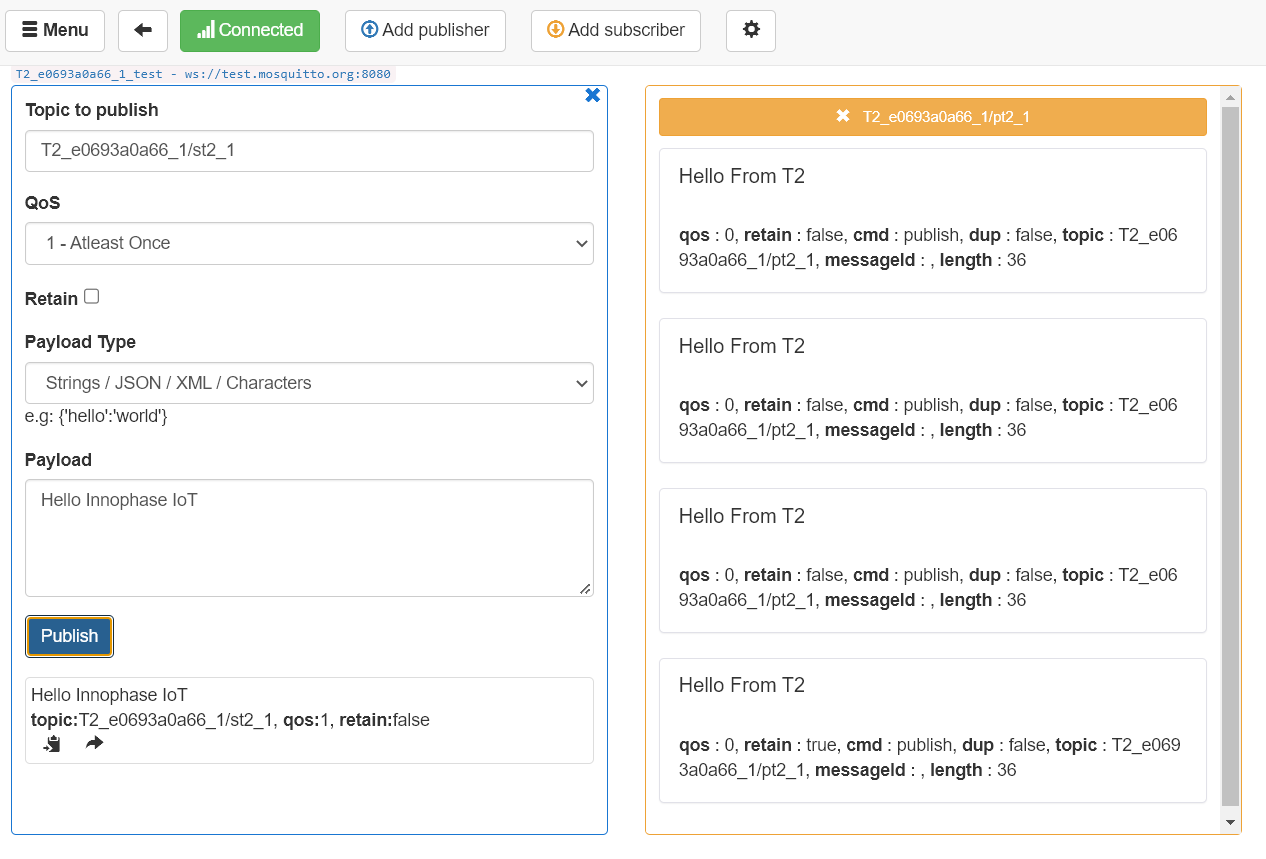


Figure : MQTTBox Extension publish subscribe window

After successfully publishing, the published message will be printed over Talaria TWO console.

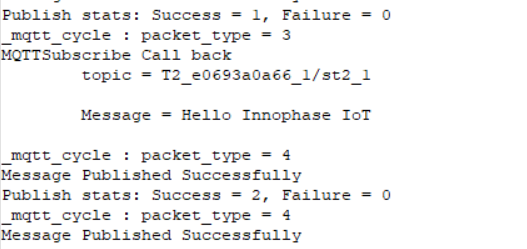


Figure : Published message over the Talaria TWO console

# Evaluating the Application using Mosquitto Local Server

## Non-secured

The application can also be evaluated by setting up a local Mosquitto server/broker on the host PC. The following steps describe the procedure to set up a local Mosquitto server and evaluate the example application.

1. Download the [mosquitto-2.0.11-install-windows-x64.exe](https://mosquitto.org/files/binary/win64/mosquitto-2.0.11-install-windows-x64.exe) from <https://mosquitto.org/download/> and install it.
2. From the command line create a password file using:

|  |
| --- |
| mosquitto\_passwd.exe -c < Name of password file > <User name> |

**Note**: Run the command prompt using administrator privileges.

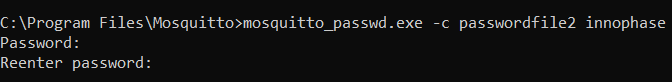


Figure : Password file generation using mosquito\_passwd

1. Append the following configuration to mosquito.conf file (open the file with administrator privileges). The allow\_anonymous trueparameter can be ignored if a password and username are used*.*

|  |
| --- |
| listener <Port number>  allow\_anonymous true  max\_keepalive <timeout>  password\_file <Path to the password file> |

as shown in the following example:

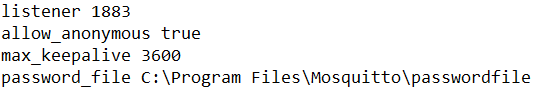


Figure : mosquito configuration file

1. Start the Mosquitto broker by issuing the following command:

|  |
| --- |
| mosquitto -c mosquitto.conf -v |

The output shown in Figure 16 will be displayed.

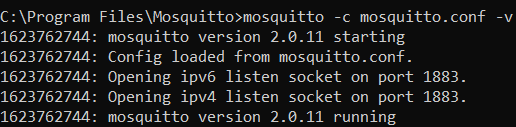


Figure 16: Starting the mosquito broker

1. Issue the following command in command prompt and check if the active connection with the listener port number is listed as shown in Figure 17 with port number 1883.

|  |
| --- |
| netstat -a |

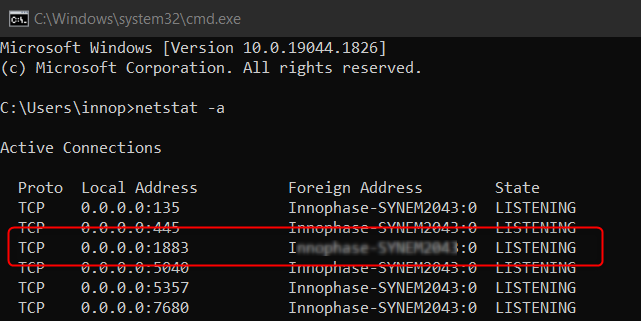


Figure : Output of netstat command

1. In another command prompt window, execute the following command to subscribe to the topic T2\_<mac id of T2\_1/pt2\_1. The MQTT server address is the IP address of the machine that is running mosquito.

|  |
| --- |
| mosquitto\_sub.exe -h <IP address of the host PC running the local MQTT server> -u <user name> -P <Password> -t T2\_<mac id of T2\_1/pt2\_1 |

In the example, the username and password are innophase and the local IP address is 192.168.1.8. Talaria TWO publishes to the topic T2\_e0693a0eba\_1/pt2\_1. Hence, the other MQTT client (PC here) will also subscribe to the same topic.

1. Program the mqtt\_non\_secured.elf along with the following bootargs using the Download Tool (refer steps from section 0 step 2).

|  |
| --- |
| cloud\_url= < IP address of the host PC running the local MQTT server>,cloud\_port=1883,cloud\_usr\_name=<user name >,cloud\_usr\_psw=<password>,mqtt\_no\_poll=1, num\_conn =1 |

1. Output is displayed in the Download Tool console, confirming that Talaria TWO is connected to the local mosquito broker running on the host PC.

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWWAE  Build $Id: git-df9b9ef $  Flash detected. flash.hw.uuid: 39483937-3207-00b0-0064-ffffffffffff  Bootargs: cloud\_url= 192.168.1.105 cloud\_port=1883 cloud\_usr\_name=innophase cloud\_usr\_psw=innophase ,mqtt\_no\_poll=1, num\_conn =1np\_conf\_path=/data/nprofile.json ssid=Lucy passphrase=Password@321  $App:git-94e4627  SDK Ver: FREERTOS\_SDK\_1.0  MQTT Example App  addr e0:69:3a:00:0a:66  Connecting to added network : Lucy  [0.801,387] CONNECT:04:d1:3a:b2:48:63 Channel:6 rssi:-35 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [0.902,070] MYIP 192.168.43.164  [0.902,233] IPv6 [fe80::e269:3aff:fe00:a66]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  Connected to added network : Lucy  app\_thread\_entry\_fnmac id:e0693a0a66  app\_auto\_generate\_params:111, size = 64  ------------------------------------------------------  T2 MQTT Client id : T2\_e0693a0a66\_1  T2 MQTT publish topic : T2\_e0693a0a66\_1/pt2\_1  T2 MQTT subscribe topic: T2\_e0693a0a66\_1/st2\_1  T2 LWT topic : will\_con1  --------------------------------------------------------  mac id:e0693a0a66  app\_auto\_generate\_params:111, size = 64  ------------------------------------------------------  T2 MQTT Client id : T2\_e0693a0a66\_2  T2 MQTT publish topic : T2\_e0693a0a66\_2/pt2\_2  T2 MQTT subscribe topic: T2\_e0693a0a66\_2/st2\_2  T2 LWT topic : will\_con2  --------------------------------------------------------  app\_mqtt\_connect 134mqtt\_init  mqtt/platform/mqtt\_nw\_tcp.c:MQTTNetworkConnect  Connecting ...  \_mqtt\_cycle : packet\_type = 2  MQTTConnect Success. ret = 0  app\_subscribe:264  \_mqtt\_cycle : packet\_type = 9  \_mqtt\_cycle : packet\_type = 5  \_mqtt\_cycle : packet\_type = 7  Message Published Successfully  Publish stats: Success = 1, Failure = 0  \_mqtt\_cycle : packet\_type = 5  \_mqtt\_cycle : packet\_type = 7  Message Published Successfully  Publish stats: Success = 2, Failure = 0  \_mqtt\_cycle : packet\_type = 5  \_mqtt\_cycle : packet\_type = 7  Message Published Successfully  Publish stats: Success = 3, Failure = 0  \_mqtt\_cycle : packet\_type = 5  \_mqtt\_cycle : packet\_type = 7  Message Published Successfully  Publish stats: Success = 4, Failure = 0  \_mqtt\_cycle : packet\_type = 5  \_mqtt\_cycle : packet\_type = 7  Message Published Successfully  Publish stats: Success = 5, Failure = 0  \_mqtt\_cycle : packet\_type = 5  \_mqtt\_cycle : packet\_type = 7  Message Published Successfully …  … |

1. Open command prompt and issue the following command:

|  |
| --- |
| mosquitto\_sub.exe -h <IP addr of local machine > -u innophase -P innophase -t T2\_e0693a0a66\_1/pt2\_1 |

1. The following sample output will be observed on the command prompt console:

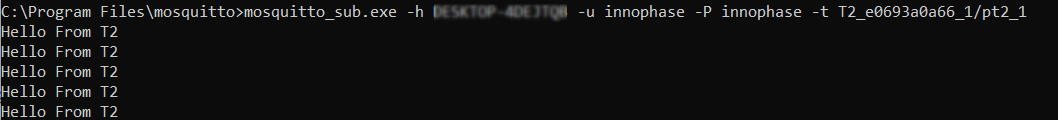


Figure : Command prompt output

The message gets published every two seconds and the count keeps incrementing.

1. In the example, Talaria TWO has subscribed to a topic T2\_e0693a0a66\_1/st2\_1. Open another command prompt window and issue the following command:

|  |
| --- |
| mosquitto\_pub.exe -h 192.168.43.3 -p 1883 -u innophase -P innophase -t T2\_e0693a0eba\_1/st2\_1 -m "Sending Data to T2" |

1. Talaria TWO received the message when the MQTT Client (host PC) published to the topic T2\_e0693a0a66\_1/st2\_1 and the output is displayed on the Download Tool’s console.

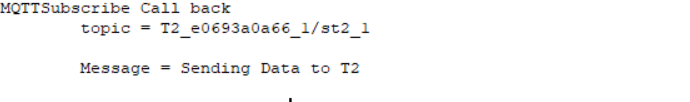


Figure 19: Download Tool Output

## Secured

The following steps describe the procedure to set up a local Mosquitto server and evaluate the example application with a secured SSL/TLS connection.

**Prerequisites**:

1. Broker: Mosquitto
2. Key and certificate generation: OpenSSL

### Key and Certificate Generation for Secured Local Server

#### Install OpenSSL

1. Download the OpenSSL v3.0.0 from the following location and install the same: <https://slproweb.com/products/Win32OpenSSL.html>.
2. Click on the Windows button on your keyboard/taskbar. Search for Environment Variables. Select Edit the system environment variables.

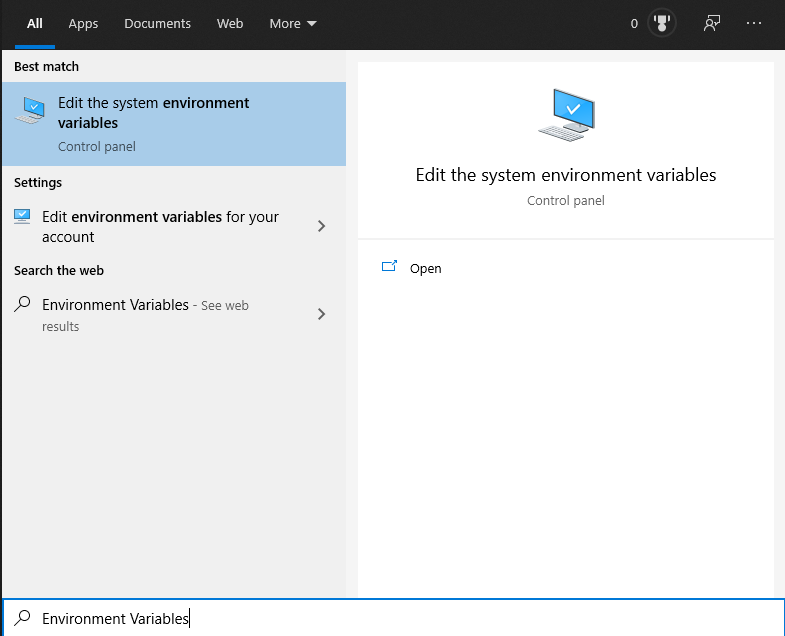


Figure : Environment variables

1. In the window that pops-up, click on Environment variables.

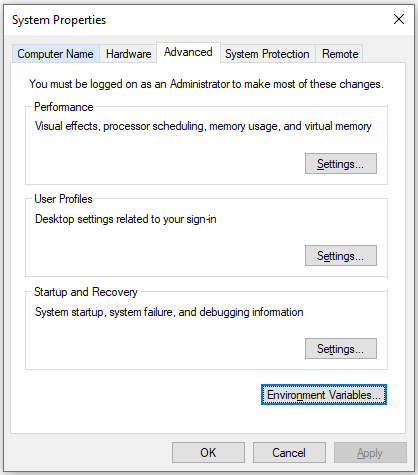


Figure : System properties

1. This pops-up a window showing User variables and System variables. In the User variables section, select Path and click Edit.

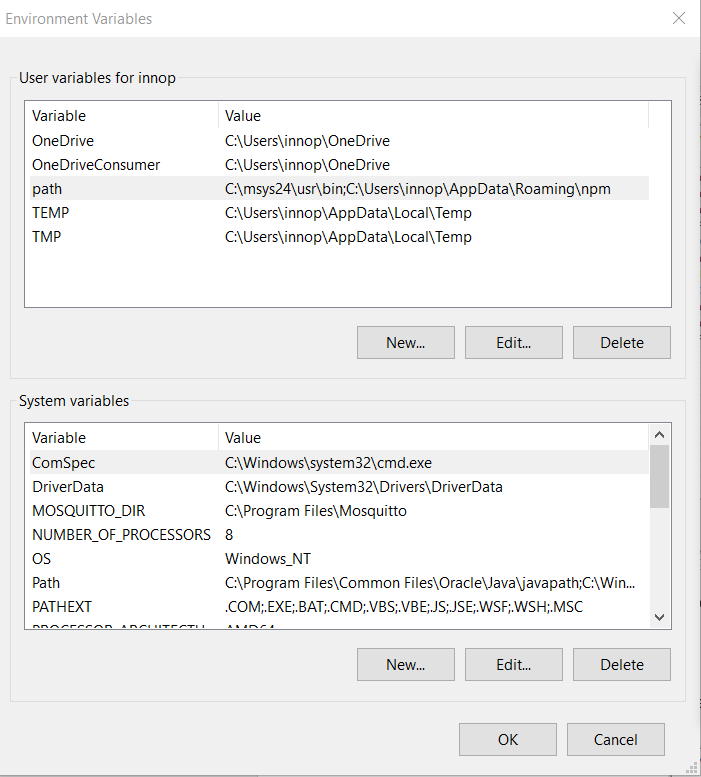


Figure : Environment variables - User variables

1. Click on Browse.

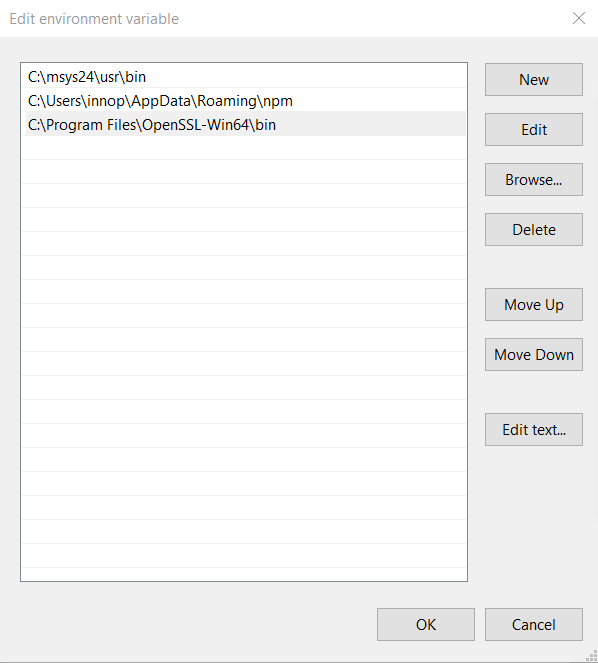


Figure : Edit Environment Variable

1. This will already have some automatically added paths for other applications. Go to where the openssl.exe is, which should be at This PC > Windows (C:) > Program Files > OpenSSL - Win64 > bin and select the folder.
2. Click OK.

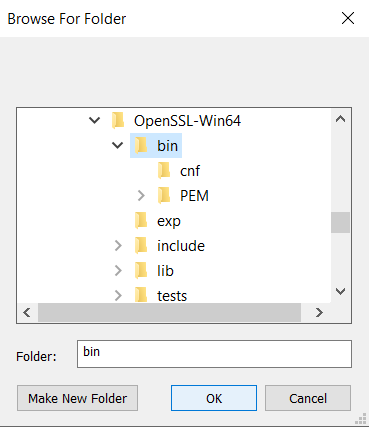


Figure : Edit Environment Variables

1. This will be added at the top. Ensure to click OK on this screen and the subsequent screens.

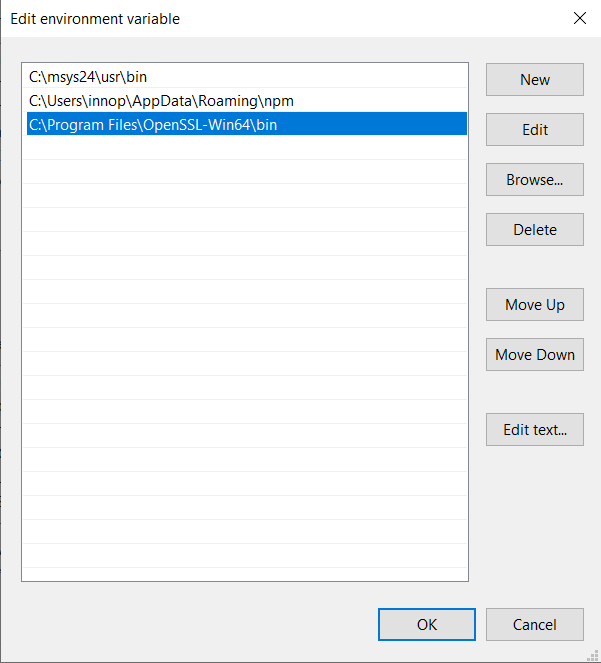


Figure : Modified Environment Variables

#### Generate Certificates

For secured Mosquitto on local, there is a need to create certificates locally using OpenSSL commands in OpenSSL command prompt. The subsequent sections provide the OpenSSL commands which can be used to generate certificates.

**Certificate Authority**

* 1. First, create a key pair for the CA using the following command on OpenSSL command prompt:

|  |
| --- |
| openssl genrsa -des3 -out ca.key 2048 |

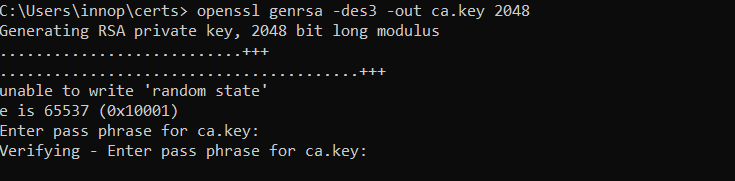


Figure : CA key generation

* 1. Create a certificate for the CA using the CA key created in step 1.

|  |
| --- |
| openssl req -new -x509 -days 1826 -key ca.key -out ca.crt |

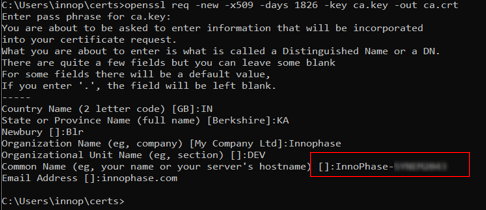


Figure : CA Certificate generation

* 1. Now we create a server key pair which will be used by the broker.

|  |
| --- |
| openssl genrsa -out server.key 2048 |

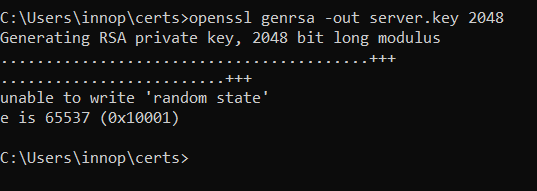


Figure : Server key generation

* 1. Now, create a certificate request .csr. When filling out the form, the common name is important and is usually the domain name of the server.

**Note:** Here, the PC name is used as a common name.

|  |
| --- |
| openssl req -new -out server.csr -key server.key |

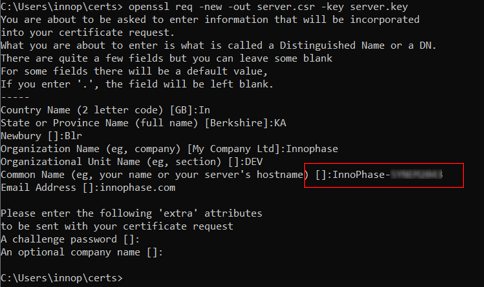


Figure : Server.csr generation

* 1. Now, the CA key is used to verify and sign the server certificate. This creates the server.crt file.

|  |
| --- |
| openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 360 |

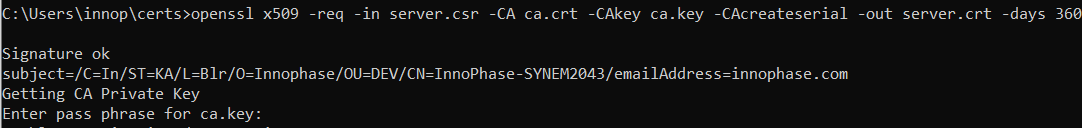


Figure : Server certificate generation

**Client Certificates**

* 1. To generate the client key, first create a client private key.

|  |
| --- |
| openssl genrsa -out client.key 2048 |

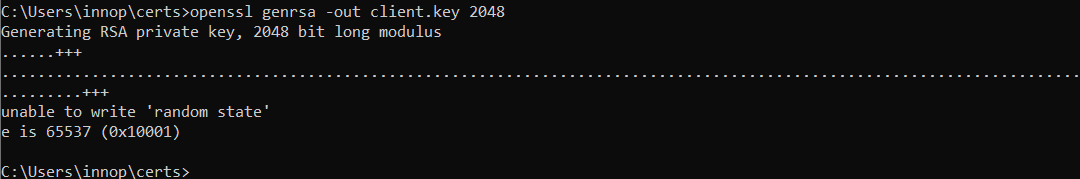


Figure : Creating client key

* 1. Next, create a certificate request and use the client’s private key to sign it.

|  |
| --- |
| openssl req -new -out client.csr -key client.key |

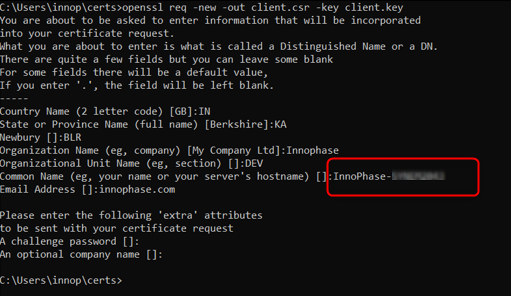


Figure : Creating client csr

* 1. Execute the following command to complete the request and create a client certificate.

|  |
| --- |
| openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client.crt -days 360 |

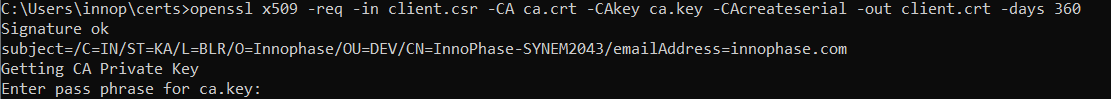


Figure : Client.crt generation

**Mosquitto Configuration File**:

|  |
| --- |
| listener <port number>  #extra listener  listener < port number >  per\_listener\_settings true  password\_file <your\_path\passwordfile>  allow\_anonymous false  cafile <your\_path\ca.crt>  certfile <your\_path\server.crt>  keyfile <your\_path\server.key>  tls\_version tlsv1.2 |

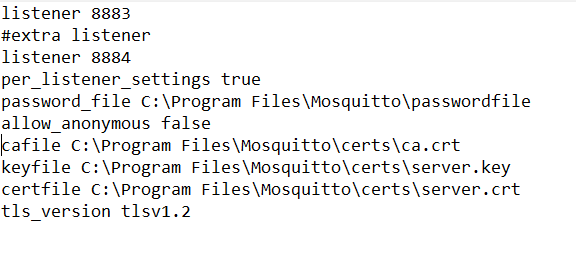


Figure : mosquitto.conf file

Start the Mosquitto broker by issuing the following command:

|  |
| --- |
| mosquitto -v -c mosquitto.conf |

Expected output:

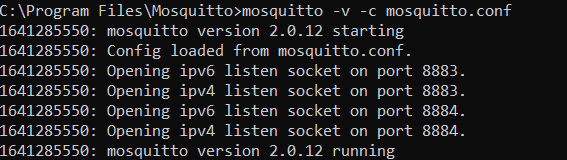


Figure : Starting the mosquito broker

Issue the following command in command prompt and check if the active connection with the listener port number is listed as shown in Figure 36 with port numbers 8884 and 8883.

|  |
| --- |
| netstat -a |

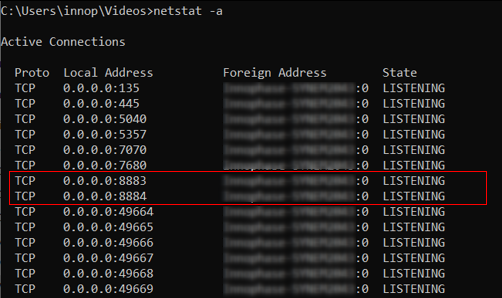


Figure : Output - netstat command

Post executing these steps, update the file system of the Talaria TWO module with newly generated certificates following the steps from section 0 step 1.

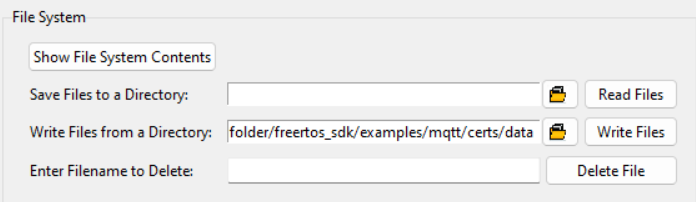


Figure : Newly generated certificates to file system

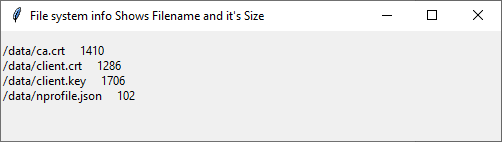


Figure : Certificates available in File System

### Programing Talaria TWO

1. Program mqtt.elf (*freertos\_sdk\_x.y/examples/mqtt/bin*)using the Download Tool.
   1. Launch the Download Tool provided with InnoPhase Talaria TWO SDK.
   2. In the GUI window:
      1. Boot Target: Select the appropriate EVK from the drop-down.
      2. ELF Input: Load the mqtt.elf by clicking on Select ELF File.
      3. AP Options: Pass the appropriate SSID and passphrase to connect to an Access Point.
      4. Programming: Prog RAM or Prog Flash as per requirement.
2. Pass the following boot arguments in the Boot Arguments field:

For secured MQTT (Verifying server certificate):

|  |
| --- |
| cloud\_url < IP address of the host PC running the local MQTT server>,cloud\_port=8883,cloud\_usr\_name=innophase,cloud\_usr\_psw=innophase,transport\_mode=1,pub\_qos=2,sub\_qos=1,ca\_cert=/data/ca.crt,mqtt\_no\_poll=1, num\_conn=1 |

For secured MQTT (No certificate verify):

|  |
| --- |
| cloud\_url=< IP address of the host PC running the local MQTT server>,cloud\_port=8883,cloud\_usr\_name=innophase,cloud\_usr\_psw=innophase,transport\_mode=3, pub\_qos=2,sub\_qos=1,mqtt\_no\_poll=1, num\_conn=1 |

For secured MQTT (Verifying the server certificate and provide the client certificate)

|  |
| --- |
| cloud\_url= < IP address of the host PC running the local MQTT server>,cloud\_port=8884,cloud\_usr\_name=innophase,cloud\_usr\_psw=innophase,transport\_mode=2,pub\_qos=2,sub\_qos=1,ca\_cert=/data/ca.crt,client\_cert=/data/client.crt,client\_key=/data/client.key,mqtt\_no\_poll=1, num\_conn=1 |

Expected Output: Local Secured MQTT:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWAE4 DWT comparators, range 0x8000  Build $Id: git-defd1fcaf $  cloud\_url=172.16.16.30 cloud\_port=8884 cloud\_usr\_name=innophase cloud\_usr\_psw=innophase transport\_mode=2 pub\_qos=1 sub\_qos=1 ca\_cert=/data/ca.crt client\_cert=/data/client.crt client\_key=/data/client.key mqtt\_no\_poll=1 num\_conn=1 np\_conf\_path=/data/nprofile.json ssid=Xiaomi\_Ax6000\_iop passphrase=InnoQA2023$  $App:git-d90a1da0  SDK Ver: FREERTOS\_SDK\_1.0  MQTT Example App  addr 02:03:04:0c:2d:50  Connecting to added network : Xiaomi\_Ax6000\_iop  [0.876,266] CONNECT:d4:da:21:54:d3:c6 Channel:13 rssi:-21 dBm  wcm\_notify\_callback :WCM\_NOTIFY\_MSG\_LINK\_UP  app\_wifi\_status\_cb: status = 757536wcm\_notify\_callback :CM\_NOTIFY\_MSG\_ADDRESS  app\_wifi\_status\_cb: status = 755720[3.262,188] MYIP 192.168.31.179  [3.262,351] IPv6 [fe80::3:4ff:fe0c:2d50]-link  wcm\_notify\_callback :\_NOTIFY\_MSG\_CONNECTED  app\_wifi\_status\_cb: status = 751512  Connected to added network : Xiaomi\_Ax6000\_iop  MQTTNoPollInit:712 - initdone = 0  MQTTRun\_NoPollThread: 658mac id:234c2d50 T2 MQTT Client id : T2\_234c2d50\_1  T2 MQTT publish topic : T2\_234c2d50\_1/pt2\_1  T2 MQTT subscribe topic: T2\_234c2d50\_1/st2\_1  T2 LWT topic : will\_con1  app\_mqtt\_connect 145  app\_mqtt\_conn\_init  mqtt/platform/mqtt\_nw\_tls.c:MQTTNetworkConnect\_Tls  . [SSL\_WRAP]Checking input configurations...  . [SSL\_WRAP]Seeding the random number generator...  . [SSL\_WRAP]Loading the CA root certificate ...Cert Len = 1221  . Loading the Client(Own) certificate ...Cert Len = 1108  . [SSL\_WRAP]Loading the Client(Own) Key ...Key Len = 1676  . [SSL\_WRAP]Connecting to tcp 172.16.16.30:8884...  . [SSL\_WRAP]Setting up the SSL/TLS structure...  . [SSL\_WRAP]setting configurations..  auth mode = 2 (0- skip, 1- optional, 2- required  max fragment len = 0  Handshake timeout = 30 Sec  . [SSL\_WRAP]Performing the SSL/TLS handshake...  . [SSL\_WRAP] Handshake done. ok  . [SSL\_WRAP]Verifying peer X.509 certificate.  Connecting ...  \_mqtt\_cycle : packet\_type = 2  MQTTConnect Success. ret = 0  app\_subscribe:295  \_mqtt\_cycle : packet\_type = 9  app\_subscribe: 299  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 1, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 2, Failure = 0  \_mqtt\_cycle : packet\_type = 4  Message Published Successfully  Publish stats: Success = 3, Failure = 0 |

Server console:

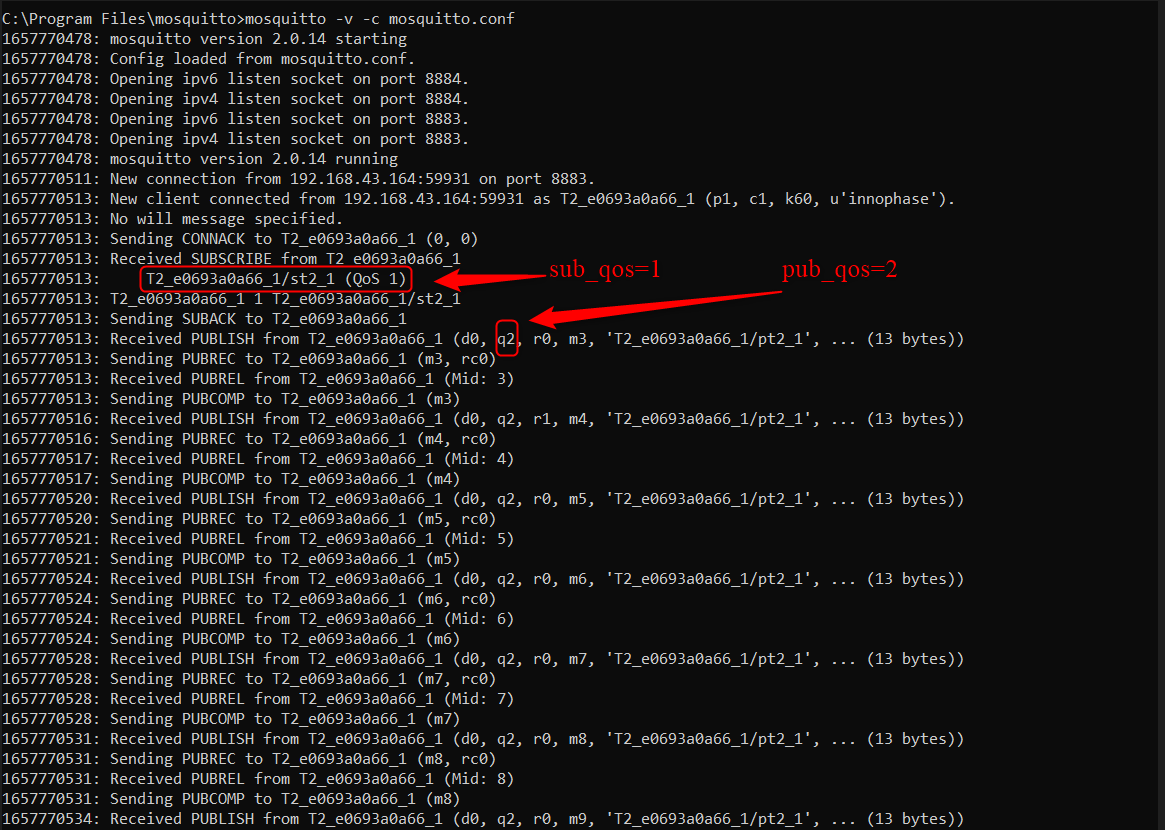


Figure : Secure MQTT - server – Console

# Certificates Validity Assessment

The certificate bundled in the MQTT example application may be expired. Ensure to check the validity of the certificate added with this example application.

Execute the following command to verify the validity of the x509 certificates:

|  |
| --- |
| cat mosquito.org.crt | openssl x509 -noout -enddate |



Figure : Certificates verification