> ***Tip***
>
> This tutorial contains functions and definitions to use the Real-Time MQTT (RT-MQTT) extention.

RT-MQTT Extention API

The presented system proposes a set of extensions to the MQTT protocol grounded on Software Defined Networking (SDN) that enable, at the network level, attaining real-time communication services. This system conveyes real-time requirements such as deadline, transmition bandwidth and flow priority to topics and end-nodes by standard MQTT messages via the user properties field and then they are used to improve the timeliness of the execution of software components at the broker and to manage the network using real-time network manager (RT-NM), creating deterministic communication channels matching the real-time requirements of the associated topics.

Components

The proposed RT-MQTT architecture is shown in Fig. 1. The system consists of the MQTT application (broker and clients), RT-NM, OF-Controller and OF-Switches components.

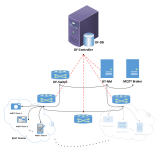

Fig.1. The architecture of RT-MQTT system.

MQTT Application

First of all, client nodes have to specify the corresponding real-time requirements via the MQTT user properties field featured by MQTT V5.0.

Mosquitto Broker

Executing the mosquitto 2.0.10 (latest version) broker to support user properties:

```
mosquitto
```

> ***Note***
>
> See also https://mosquitto.org.

MQTT Clients

Creating the MQTT clients defined real-time MQTT user properties:

a. Real-time MQTT user properties:

- Dead-line (Second): It used to enables forwarding and scheduling decisions for time critical MQTT applications.

- Minimum and maximum bandwidth (Mbps): It defines how much data can be sent over a specific connection in an amount of time.
- Flwo priority (int value): It assigns different priority to assess the effectiveness of time-sensitive traffic prioritization.

b. MQTT messages carried away properties:

- CONNECT Message: Setting the real-time requirements initially, during the connection phase.
- PUBLISH Message: Updating the real-time requirements, during the publishing phase.
- SUBSCRIBE Message: Specifying the real-time requirements, during the subscribing phase.

c. Executing the user properties:

i.Import properties function from MQTT V5.0:

```
from paho.mqtt.properties import Properties
```

ii.Creating clients:

```
client = mqtt.Client("client",protocol=mqtt.MQTTv5)
```

Parameters:

- client: Getting the client class.
- protocol: Specifying the MQTT version.

iii.Call back for the CONNECT acknowledgement:

```
def on_connect(client, userdata, flags, reasonCode, properties=None):
print('Connected properties', properties)
```

Parameters:

- client: Defining the client instance for this callback.
- userdata: It is the private user data as set in Client() or user_data_set().
- flags: It is response flags sent by the broker.
- reasonCode: It is used for checking that the connection was established.
- properties: Defined an array of user defined UTF-8 key/value pairs.

Note: Flags and userdata aren't normally used.

iv.Setting the CONNECT function user properties:

```
properties_con=Properties(PacketTypes.CONNECT)
properties_con.deadline = d       #set deadline time
properties_con.min_bw = mb        #set minimum bandwidth
properties_con.max_bw = Mb        #set maximum bandwidth
properties_con.flwo_priority = p  #set flow priority

client_pub.connect(host, port, keepalive, properties=properties_con)
```

Parameters:

- d: String to define a value as a deadline packet time.
- mb: String to define a value as a minumum data rate bandwidth.

- Mb: String to define a value as a maximum data rate bandwidth.
- p: String to define a value as a flwo priority.
- host: It is the hostname or IP address of the remote broker.
- port: It is the network port of the server host to connect to.
- keepalive: It is a maximum period in seconds allowed between communications with the broker.
- properties: Defined an array of user defined UTF-8 key/value pairs.

v.Call back for the PUBLISH acknowledgement:

```
def on_publish(client, userdata, mid):
print("Payload is published")
```

Parameters:

- client: Defining the client instance for this callback.
- userdata: It is the private user data as set in Client() or user_data_set().
- mid: The mid variable matches the mid variable returned from the corresponding publish() call, to allow outgoing messages to be tracked.

vi.Setting the PUBLISH function user properties:

```
properties=Properties(PacketTypes.PUBLISH)
properties.UserProperty=[("properties_con.deadline = d", "properties_con.min_bw = mb", "properties_con.max_bw = Mb", "properties_con.flwo_priority = p")]

client_pub.publish(pub_topic, payload, qos, properties=properties)
```

Parameters:

- d: String to define a value as a deadline packet time.
- mb: String to define a value as a minumum data rate bandwidth.
- Mb: String to define a value as a maximum data rate bandwidth.
- p: String to define a value as a flwo priority.
- pub_topic: It is the topic that the message should be published on.
- payload: It is the actual message to send.
- qos: It is the quality of service level to use.
- properties: Defined an array of user defined UTF-8 key/value pairs.

vii.Call back for the SUBSCRIBE data:

```
def on_subscribe(client, userdata, mid, granted_qos,properties=None):
    print('SUBSCRIBED')
```

Parameter:

- client: Defining the client instance for this callback.
- userdata: It is the private user data as set in Client() or user_data_set().
- mid: The mid variable matches the mid variable returned from the corresponding publish() call, to allow outgoing messages to be tracked.
- granted_qos: The granted_qos variable is a list of integers that give the QoS level the broker has granted for each of the different subscription requests.
- properties: Defined an array of user defined UTF-8 key/value pairs.

RT-NM

The RT-NM intercepts all messages directed to the MQTT broker. It extracts real-time user properties (deadline, bandwidth and priority) and updates the OF-DB tables. As illustrated in Fig. 2, these parameters then communicated to the OF-Controller for updating the OF-Switches flow tables.

Fig.2. RT-NM operation.

As an example of message sequence chart, the processing of a PUBLISH or SUBSCRIBE message is sketched in Fig. 3.

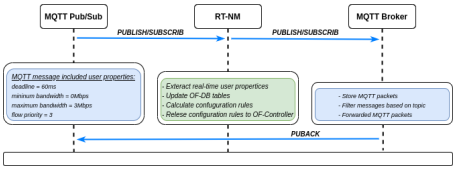

Fig.3. Real-time attributes conveyed in a PUBLISH/SUBSCRIBE message.

Functions

i.Defining clients class:

```
client = mqtt.Client("client",protocol=mqtt.MQTTv5)
```

Parameters:

- client: Getting the client class.
- protocol: Specifying the MQTT version.

ii.Defining call-back function that will runned whenever there is a message (payload) published on the given topic. This function extracts details from the msg user properties:

```
def on_message(client, msg):
```

Parameters:

- client: Getting the client class.
- msg: It is a published message details received from the client.

iii.Updating the OF-DB tables using user properties:

```
rcv_msg = msg.payload.decode()
idx = rcv_msg.find('#*')
```

Parameter:

- rcv_msg: rcv_msg could be a tuple form includs (“<topic>”, “<payload>”, qos, retain).

OF-Controller

Runing the OF-Controller.

```
ryu-manager --verbose --observe-links ~/flowmanager/flowmanager.py ryu_controller.py rest_qos.py rest_conf_switch.py
```

Definitions

- ryu_controller.py: Ryu framwork which is able to support bandwidth reservation.
- rest_qos.py: The rest_qos.py is supposed to be processed on Flow Table pipeline processing, modify ryu_controller.py to register flow entry into table id:1.
- rest_conf_switch.py: This module provides a set of REST API for switch configuration.

Arguments

- ryu-manager: It listens to ip address 0.0.0.0 and port 6633 by default to connect to openflow switch (hardware or openvswitch OVS).
- --observe-links: It is used to observe link discovery events.
- --verbose: It shows the debug output.

> ***Note***
>
> Check out more information on https://ryu-sdn.org/.

OF-Switch

OF-Controller updates the OF-Switches flow tables based on time-sensitive parameters and configurs them to create deterministic communication channels matching the received real-time parameters of the associated topics.

Adding bredge, ports and queue

Creating bridge, port and adding queue to transfer normal and time sensitive data packets.

i.Creating a bridge on the switch:

```
ovs-vsctl add-br <bridge_name> [-- set bridge br0 datapath_type=pica8]".
```

Parameter

- bridge_name: Bridge name id.

ii.Adding access ports to created bridge:

```
ovs-vsctl add-port br0 <port> vlan_mode=access tag=1 -- set Interface <port> type=pica8
```

Parameter

- <port>: Ethernet switching port identifier such as ge-1/1/1.

iii. Setting the port link speed, mtu and :

ovs-vsctl add-port br0 ge-1/1/1 vlan_mode=access tag=1 -- set Interface ge-1/1/1 type=pica8 options:link_speed=<speed-value>,mtu=<mtu-value>,flow_ctl=<flow-value>

Parameters

- <speed-value>: Port speed identifier as 100M (Mbps).
- <mtu-value>: Maximum transmit packet size for a specified port as 1500.
- <flow-value>: Flow control value support tx, rx, tx_rx and none.

iv. Creating a qos and add one queue (0) to qos for port ge-1/1/1:

ovs-vsctl -- set port ge-1/1/1 qos=@newqos -- --id=<qos_id> create qos type=PRONTO_STRICT queues:0=<queue_id> -- --id=@newqueue0 create queue other-config:min-rate=<min-value> other-config:max-rate=<max-value>

Parameters

- <qos_id>: Setting QoS id for created port as @newqos.
- <queue_id>: Setting queue id for creted port as 0=@newqueue0.
- <min-value>: Minimum bandwidth rate for creted port as 1000 (bps).
- <max-value>: Maximum bandwidth rate for creted port as 3000 (bps).

v. Configuring match fields and setting actions:

```
ovs-ofctl add-flow br0 priority=<pr_value>,in_port=<in_port>,dl_type=<dl_type>,nw_proto=<nw_proto>,nw_src=<nw_src>,nw_dst=<nw_dst>,tp_src=<tp_src>,tp_dst=<tp_dst>,actions=set_queue:<ENQUEUE>,output=<OUTPUT>,goto_table:<GOTO_TABLE>
```

Parameters

- <pr_value>: Flow table entry priority.
- <<in_port>>: Input switch port (int).
- <dl_type>: Ethernet frame type (int).
- <nw_proto>: IP protocol or lower 8 bits of ARP opcode (int).
- <nw_src>: IPv4 source address (string).
- <nw_dst>: IPv4 destination address (string).
- <tp_src>: TCP/UDP source port (int).
- <tp_dst>: TCP/UDP destination port (int).
- <ENQUEUE>: Output to queue with "queue_id" attached to "port".
- <OUTPUT>: Output packet from "port"
- <GOTO_TABLE>: Instruction) Setup the next table identified by "table_id".

vi. Viewing the bridge settings:

ovs-ofctl show <bridge_name>

Parameter

- <bridge_name>: Bridge name id

Guide