ACKSYS
COMMUNICATIONS & SYSTEMS

APPLICATION NOTE

APNUS0041 How To Monitor & Configure ACKSYS Router via MQTT

September 2024

Content

1. MQTT Glossary and Term

MQTT – Message Queuing Telemetry Transport.

MQTT Broker – The central server to which MQTT client connect in charge of managing message topics.

MQTT Client – All devices and software, such as the ACKSYS Router or WaveManager 4.0.1, that are connected to the broker

MQTT topic- a UTF-8 encoded string that is the basis for message routing in the MQTT protocol

MQTT payload - Messages shared with other devices or software via a broker using MQTT. Data format agnostic and can support anything including text, images, binary numbers, etc.

JSON – JavaScript Object Notation, a lightweight data-interchange format.

SERV_ID – Service Identification.

DEV_ID – MotherBoard Identification.

Publish – When a client publishes to a topic on the MQTT broker, it is updating the data associated with that topic on the broker, and publishing new messages for the topic's subscribers (if any)

Subscribe –Once a client subscribes to a topic on the MQTT broker, it will receive all of the subsequent messages that have been published to the topic

–The multi-level ('#') wildcard used to specify all remaining levels of hierarchies and must be the last character in the topic subscription string

AWS –Amazone Web Services

2. Introduction

Monitoring a park of router become more and more crucial for operator for IOT and IIoT through MQTT.

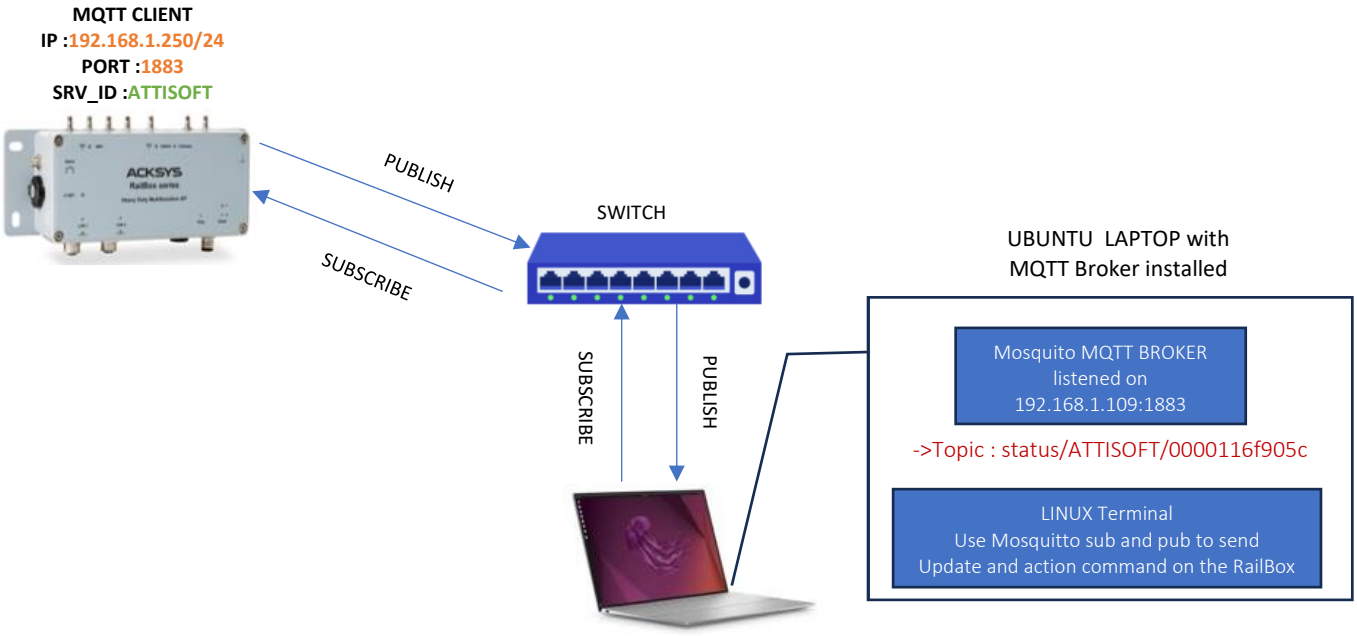
MQTT (Message Queuing Telemetry Transport) is a messaging protocol for restricted low-bandwidth networks and extremely low-latency to connect IoT devices. Queuing Telemetry Transport is implementation in WaveOs for:

- Data collection
- Real-time monitoring
- Product configuration and management
- Firmware upgrade
- Network topology discovery

In this application note, we will explain in detail the basic steps required to configure Acksys Router as MQTT client to connect to a local Broker (on premise) installed on the LAN.

3. On Premise MQTT Architecture

In this application note, we will use 1 Acksys Router as MQTT client and MQTT Broker installed on a PC running on Ubuntu and not on ACKSYS Cloud (hosted on AWS).



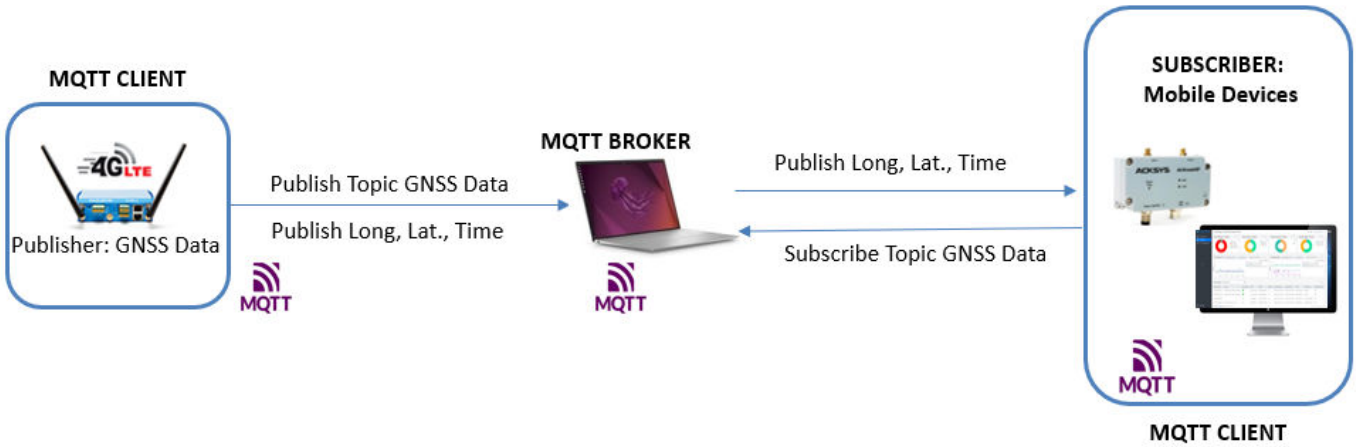
Before we begin, let's overview the configuration that we are attempting to achieve and the prerequisites that make it possible in this application note :

- 1 RailBox router or Any type of Acksys Router in release 4.26.0.1 or Higher
- A switch to connected the MQTT Client (Router) and the Broker
- A Linux PC on which the Broker is installed

How MQTT Works MQTT Publish / Subscribe Architecture

MQTT is based on a publish/subscribe model.

*Devices (**MQTT clients**) publish data (**Topics**) to a server (**MQTT broker**) and other devices or systems (MQTT clients) subscribe to topics to receive data.*



*The main points to consider about **MQTT** are:*

MQTT clients:

- *they can publish messages on topics (e.g., "GNSS/from/AirBox")*
- *they receive messages from subscribed topics (e.g., "GNSS/from/AirXroad") via the MQTT broker*
- *they keep the connection to the broker established or reconnect in case of connection loss*

MQTT brokers:

- *They serve as the central hub in the publish/subscribe messaging system*
- *They receive published messages and dispatch the message to the subscribing MQTT clients.*
- *An MQTT message contains a message topic that MQTT clients subscribe to*

4. ACKSYS Router configuration

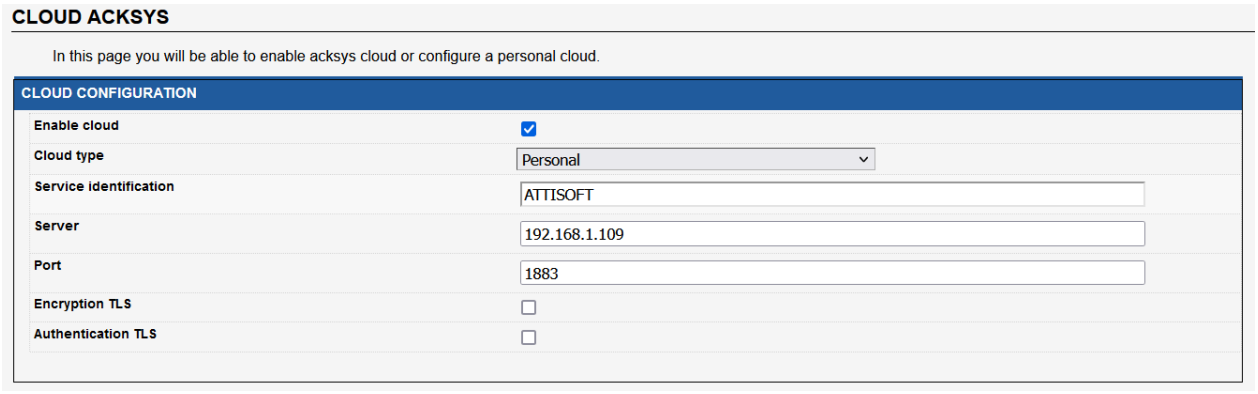
Let keeping in mind that all Acksys routers are compatible with MQTT Broker only in its version 4.26.0.1 of higher with a predefined list of Topics.

Configuring MQTT Client

If you have familiarized yourself with the configuration scheme, we can start configuring the router using instructions provided.

LAN Network	MQTT Client	MQTT Broker
192.168.1.0/24	RailBox:192.168.1.250/24	Ubuntu: 192.168.1.106/24

In the GUI, go to Setup → Service → Cloud to configure the MQTT client to interact with the MQTT Broker installed on LAN Network with the following information.



Field name	Sample	Description
Enable	Checked	Enable MQTT service
Cloud Type	personal	Personal Cloud
Service identification	ATTISOFT	This field is mandatory. Provided by the web interface when cloud is personal.
Server	192.168.1.106	Remote Broker's address
Port	By default :1883	Select which port the broker should use to listen for connections
Encryption TLS	Unchecked	Enable TLS/SSL authentication for the broker
Authentication TLS	Unchecked	Enable TLS/SSL authentication for the broker

NOTE: The device identification dev_id, formatted version of the device hard id, and the service identification serv_id, provided by the user on the web UI, set the topics.

5. ACKSYS Predefined MQTT Topics

There is a predefined list of MQTT topics used on the Broker (the server) to filter messages for connected clients. Topics may consist of one to multiple levels.

This is an extract of ACKSYS MQTT interface. For complete documentation on ACKSYS MQTT parameters please refer to the ACKSYS Userguide.

(This topics list maybe updated in the future, please ensure that you are using the latest Userguide version.)

Topics list

Field name	Sample	Description
Configuration state	configuration/state/serv_id/dev_id/request configuration/state/serv_id/ dev_id	-The configuration state allows to monitor any change made to the product. It is a hash of the current static configuration. -The hash can be done on all the config or on a specific configuration data. -The configuration is sent by the product to the broker in the following cases: ▪ Bootup ▪ Configuration change ▪ Upon request
Configuration data	configuration/data/serv_id/dev_id /request configuration/data/serv_id/ dev_id	Once you detect a new product or a configuration hash change, you can request the associated configuration data to review the elements which have changed. The request is made in a JSON format by providing a payload to the request (a table containing the config_id of the elements you want to retrieve): Ex: 'ExpectedConfigurationData' : [{' SystemConfiguration'}, {' WirelessConfiguration'}]

Telemetry data	status/serv_id/dev_id	The products collects periodically (with a configurable interval) the telemetry statistics.
		The telemetry data are sent using the JSON format:
		Ex:
		{
		"ProductId" : "router_id",
		"MessageOrder" : "order",
		"Data" :[
		{"Type" : 1, bandwidth_data},
		{"Type" : 2, wireless_data},
		{"Type" : 3, roaming_data},
		{"Type" : 7, cellular_data}
		]
		}

Updates	command/update/serv_id/dev_id/request command/update/serv_id/dev_id	The update commands can be used for: ▪ FirmwareUpdate = 1, ▪ ConfigurationFileUpdate = 2, ▪ SystemUpdate = 3,

		▪ PhysicalInterfaceUpdate = 4, ▪ WirelessUpdate = 5, ▪ NetworkUpdate = 6, ▪ WebServerUpdate = 7, ▪ StatusSettingUpdate = 8 ▪ ResetToFactory=9 The update can be done on one or multiple configurations at the same time
Actions	command/action/serv_id/dev_id /request command/action/serv_id/dev_id	The action commands can be used for: ▪ ConfigurationFileDownloadAction = 1, ▪ LedTrackingAction = 2, ▪ WifiScanAction = 3, ▪ PingAction = 4, ▪ TracerouteAction = 5, ▪ BandwidthTestAction = 6, ▪ DNSTestAction = 7, ▪ Reboot = 8 (coming in 4.28.0.1) ▪ ResetToFactory=9 (coming in 4.28.0.1)

Configuration State: The state is sent in a JSON format, with the following data:

Element	Type/Format
ProductId	String
Code	Int
SerialNumber	String / NNNNNNNN
DefaultIPAddress	String / N.N.N.N or String / XXXX:XXXX:XXXX:XXXX:XXXX:XXXX:XXXX:XXXX
GlobalConfigurationHash	String
SystemConfigurationHash	String
PhysicalInterfaceConfigurationsHash	String
CapabilityConfigurationHash	String
WirelessConfigurationsHash	String
NetworkConfigurationsHash	String
StatusServiceConfigurationHash	String
ProtocolVersion	String

Configuration Data : Config_ids that are available

Component	Identifier
Complete configuration	AllConfigurations
System configuration	SystemConfiguration
Physical interface configuration	PhysicalInterfaceConfigurations
Wireless configuration	WirelessConfigurations
Wireless capabilities by country	CapabilityConfiguration
Network configuration	NetworkConfigurations
Static service configuration	StatusServiceConfiguration

Physical interface configuration Update:

Element	Type	Possible values
Name	String	radio0, radio1, eth0, eth1, wwan0, gps0, gps1, etc.
Type	Int/Enumeration	Unknown = 0, Ethernet = 1, Wireless = 2, Cellular = 3, GPS = 4
Label	String	
State	Int/Enumeration	Disabled = 0, Enabled = 1
Band	Int/Enumeration	Unknown = 0, Band_24 = 1, Band_5 = 2, Band_6 = 3
WifiMode	Int/Enumeration	None = 0, … AC = 8, AX = 9
ChannelWidth	Int/ Enumeration	None = 0, HT20 = 1, ….HE40 = 11, HE80 = 12, HE80_80 = 13, HE160 = 14
PrimaryWideSegment	Int/Enumeration	Channel_36 = 36, Channel_40 = 40, Etc.
PrimaryChannel	Int/ Enumeration	Auto_Channel = 0, Channel_36 = 36, Channel_40 = 40, Etc.
SecondaryWideSegment	Int/Enumeration	Channel_36 = 36, Channel_40 = 40, Etc.
MaximalTransmitPower_dBm	Int	
MTU_O	Int32	
MACAddress	String/XX:XX:XX:XX:XX:XX	

TELEMETRY: List of available statistics :

Identifier	Enumeration code
BandwidthStatus	1
WirelessStatus	2
RoamingStatus	3
AssociationStatus	4
MeshLinkStatus	5
GPSStatus	6
CellularStatus	7
SystemStatus	8
NetworkStatus	9

TELEMETRY Example: Data for association state:

Element	Type
WirelessInterfaceName	String
DateTime_UTC	Int64
Ssid	String
BSsid	String
Channel	Int
Signal_dBm	Int
SignalQuality	Int
Noise_dBm	Int
SNR	Int
ClientMACAddress	String / XX:XX:XX:XX:XX:XX

BandWidth Data:

Element	Description	Type	Possible Values
InterfaceName	Name of Interface	String	wlan0, wlan0_1, etc.
InterfaceType		Int/numeration	Unknown = 0, Ethernet = 1, Wireless = 2, Cellular = 3, Network = 5
DateTime_UTC	Date and time of the record :In unix timestamp in ms	Int64	
RXSpeed_Bs	Inbound throughput in bytes/ second	Int	String

TXSpeed_Bs	Outgoing throughput in bytes/second	Int	Int
RXProcessed_Ps	Name of packets incoming processes/second	Int	Int
TXProcessed_Ps	Name of packets outgoing treaties/second	Int	Int
RXDropped_Ps	Name of packets abandoned entering by second	Int	Int
TXDropped_Ps	Name of packets abandoned by second	Int	Int
RXError_Ps	Name of packetsErroneous Entering by second	Int	String / XX:XX:XX:XX:XX:XX
TXError_Ps	Name of packets erroneous outgoing by second		
RXError_Ps	Name of packets Erroneous Medium Input/second	double	
TXError_Ps	Name of packets Erroneous outgoing average/second	double	

System Date:

Element	Description	Type	Possible Values
HorlogeState	Name of Interface	Int/numeration	NotSynchronized = 0, Synchronized = 1
HorlogeSynchronization Method	Method of synchronizing The clock	Int/numeration	None = 0, NTP = 1, Cellular = 2

Cellular Data:

Element	Description	Type	Possible Values
PhysicalInterfaceName	Name of the associated cellular physical interface	String	wwan0
DateTime_UTC	Date and time of registration: in Unix timestamp in milliseconds	Int64	
CurrentSIM	SIM card used	Int/numeration	Card1 = 1, Card2 = 2, None = 255
Signal_dBm	Niveau de signal en dBm	Int	
DataCount	Taille de données	Int64	
RegisterState	Registration Status	Int/numeration	
SIMSwitchMode	SIM Card Switching Mode	Int/numeration	
Operator	Name of the telecom operator	String	
IMEI	International Mobile Equipment Identity Cellular Interface	String	
CellularModel	Cellular Interface Model	String	

MACAddress	Physical address	String /XX:XX:XX:XX:XX:XX	

Network Data:

Element	Description	Type	Possible Values
NetworkInterfaceName	Name of Interface	String	Lan,….
IPv4Gateway	IPV4 Gateway	String / N.N.N.N	
MACAddress	Physical Address	String /XX:XX:XX:XX:XX:XX	
IPv4Addresses	IPv4 Address	[{"IPv4Address":"String/N.N.N.N,"IPv4Mask": String /N.N.N.N}]	
IPv4DNSAddresses	DNS IPv4 Address	[String / N.N.N.N]	
IPv6DNSAddresses	IPv6 Address	[{"IPv6Address":String/XX:XX:XX:XX:XX:XX,"IPv6PrefixLength": Int, "Type": Int }]	

Association Data:

Element	Description	Type	Possible Values
WirelessInterfaceName	Name of Wireless Interface	String	Radio0W0, etc..,….
DateTime_UTC	Date and time of Registration: in Unix timestamp in Milliseconds	Int64	
Ssid	Service Set Identifier	String	
BSsid	Basic Service Set Identifier	String	
Channel	Channel used	Int	-1 channel indicates that the product is searching for a channel
Signal_dBm	IPv6 Address	Int	N/A
SignalQuality		Int	
Noise_dBm			
SNR		Int	
ClientMACAddress		String/ XX:XX:XX:XX:XX:XX	ClientMACAddress

6. Configuring MQTT Broker

To test ACKSYS MQTT integration, we will use a Ubuntu laptop on which is installed Eclipse Mosquitto, as Broker. A lightweight communication protocol based on the publisher/subscriber concept as an alternative to the classic client/server architecture, widely used in the Internet of Things.

NOTE: In this application note, we will not describe in detail how to install and configure Mosquitto software (MQTT Broker), on Ubuntu Server but how to use it to test the exchange of messages between Publisher and Subscriber.

Once Mosquitto is installed, it provides the mosquitto_pub and mosquitto_sub command line MQTT clients which you can use to perform testing or troubleshooting, carried out manually or scripted.

Let checking if Mosquito is active and running as expected as service with the following command: `sudo systemctl status mosquitto`

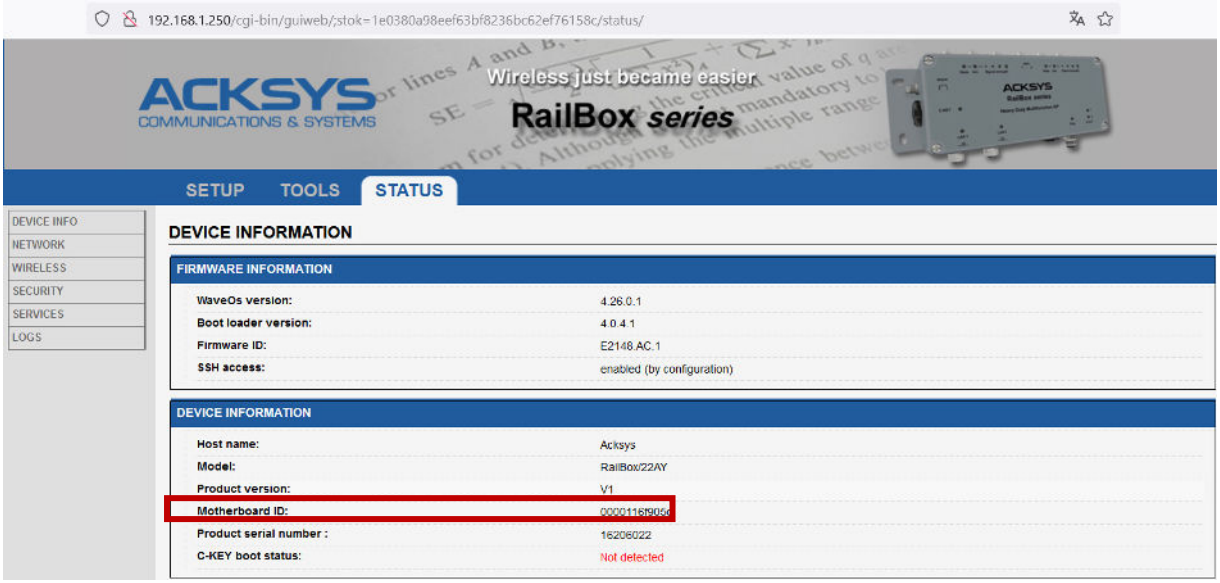
```
ubuntu@ubuntu-Attisoft:~$ sudo systemctl status mosquitto
[sudo] Mot de passe de ubuntu :
 mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
     Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2024-08-06 15:26:46 CEST; 46min ago
       Docs: man:mosquitto.conf(5)
             man:mosquitto(8)
   Main PID: 8471 (mosquitto)
      Tasks: 3 (limit: 2293)
     Memory: 1.7M
     CGroup: /system.slice/mosquitto.service
             └─8471 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

août 06 15:26:46 ubuntu-Attisoft systemd[1]: mosquitto.service: Succeeded.
août 06 15:26:46 ubuntu-Attisoft systemd[1]: Stopped Mosquitto MQTT v3.1/v3.1.1 Broker.
août 06 15:26:46 ubuntu-Attisoft systemd[1]: Starting Mosquitto MQTT v3.1/v3.1.1 Broker...
août 06 15:26:46 ubuntu-Attisoft mosquitto[8471]: [  939.381226]~DLT~ 8471~INFO     ~FIFO /tmp/dlt cannot be opened.
Retry>
août 06 15:26:46 ubuntu-Attisoft systemd[1]: Started Mosquitto MQTT v3.1/v3.1.1 Broker.
```

If Mosquitto isn't already active, enable it manually by typing: `sudo systemctl start mosquito`
The broker is opened it port to accept the MQTT client connection as shown the below screenshot to read the log.

```
ubuntu@ubuntu-Attisoft:~$ sudo tail -f /var/log/mosquitto/mosquitto.log
[sudo] Mot de passe de ubuntu :
1722938432: Client mosq-6922KpHgEpkaqKrZVf disconnected.
1722938732: mosquitto version 1.6.9 terminating
1722938732: Saving in-memory database to /var/lib/mosquitto/mosquitto.db.
1722938742: mosquitto version 1.6.9 starting
1722938742: Config loaded from /etc/mosquitto/mosquitto.conf.
1722938742: Opening ipv4 listen socket on port 1883.
1722938742: Opening ipv6 listen socket on port 1883.
1722938743: New connection from 192.168.1.250 on port 1883.
1722938743: New client connected from 192.168.1.250 as 0000116f905c (p2, c1, k60).
1722938748: New connection from 127.0.0.1 on port 1883.
```

The messages transit through the MQTT network on Topics, the identification paths of the messages. Often, these paths are also organized hierarchically (ex: **/status/serv_id/dev_id where** dev_Id is the motherboard ID of the RailBox Router).



7. TESTING

If you've followed all the steps presented above, your configuration should be finished. But as with any other configuration, it is always wise to test the setup in order to make sure that it works properly.

Test on subscribing for telemetry status Topic

The MQTT network provides for the presence of 3 main fundamental components: Broker - Publisher - Subscriber. Mosquitto itself acts as a Broker, i.e. the one in charge of redirecting messages between the relevant senders and recipients.

- **MQTT client subscribing on the MQTT Broker:**

  Here the MQTT client will run on the same laptop than the broker. It will consist on a the call of mosquito_sub cmd. We can see a JSON formatted chain containing the product telemetry datas.

```
ubuntu@ubuntu-Attisoft:~$ mosquitto_sub -t "status/ATTISOFT/0000116f905c"
{"ProductId":"0000116f905c","MessageOrder":98,"Data":[{"Type":1,"InterfaceType":1,"InterfaceName":"eth0","DateTime_UTC
":1534407952478,"RXSpeed_Bs":1180,"TXSpeed_Bs":1288,"RXProcessed_Ps":12,"TXProcessed_Ps":12,"RXDropped_Ps":0,"TXD
ropped_Ps":0,"RXError_Ps":0,"TXError_Ps":0},{"Type":1,"InterfaceType":2,"InterfaceName":"radio1w0","DateTime_UTC":1534
407952478,"RXSpeed_Bs":0,"TXSpeed_Bs":0,"RXProcessed_Ps":0,"TXProcessed_Ps":0,"RXDropped_Ps":0,"TXDropped_Ps":0,"R
XError_Ps":0,"TXError_Ps":0},{"Type":1,"InterfaceType":2,"InterfaceName":"radio0w0","DateTime_UTC":1534407952478,"RXS
peed_Bs":0,"TXSpeed_Bs":0,"RXProcessed_Ps":0,"TXProcessed_Ps":0,"RXDropped_Ps":0,"TXDropped_Ps":0,"RXError_Ps":0,"TX
Error_Ps":0},{"Type":1,"InterfaceType":1,"InterfaceName":"eth1","DateTime_UTC":1534407952478,"RXSpeed_Bs":0,"TXSpeed
_Bs":0,"RXProcessed_Ps":0,"TXProcessed_Ps":0,"RXDropped_Ps":0,"TXDropped_Ps":0,"RXError_Ps":0,"TXError_Ps":0},{"Type":
8,"HorlogeState":1,"HorlogeSynchronizationMethod":1}]}
```

As expected, we receive data every 10 seconds for status Topics from the RailBox as MQTT Client.

Log from MQTT Client:RailBox

For info, you can see the mqtt msg run by the Railbox product in the log system. The Railbox will push the telemetry data to the broker.

```
root@Acksys:~# logread -f
Thu Aug 16 08:27:52 2018 daemon.emerg : mqttd: Client 0000116f905c sending PUBLISH (d0, q1, r0, m118,
'status/ATTISOFT/0000116f905c', ... (985 bytes))
Thu Aug 16 08:27:52 2018 daemon.emerg : mqttd: Client 0000116f905c received PUBACK (Mid: 118, RC:0)
Thu Aug 16 08:28:00 2018 cron.info crond[21551]: USER root pid 1894 cmd /usr/sbin/ack_service/ack_service_check
```

```
Thu Aug 16 08:28:00 2018 cron.info crond[21551]: USER root pid 1895 cmd /usr/sbin/acksys_telemetry_check
Thu Aug 16 08:28:02 2018 daemon.emerg : mqttd: Client 0000116f905c sending PUBLISH (d0, q1, r0, m119, 'status/ATTISOFT/0000116f905c', ... (984 bytes))
Thu Aug 16 08:28:02 2018 daemon.emerg : mqttd: Client 0000116f905c received PUBACK (Mid: 119, RC:0)
Thu Aug 16 08:28:12 2018 daemon.emerg : mqttd: Client 0000116f905c sending PUBLISH (d0, q1, r0, m120, 'status/ATTISOFT/0000116f905c', ... (983 bytes))
Thu Aug 16 08:28:12 2018 daemon.emerg : mqttd: Client 0000116f905c received PUBACK (Mid: 120, RC:0)
```

8. Example of some action commands

For better understanding of MQTT implemented in WaveOs, please find below the example list of action command available:

Enable Led Tracking with the MQTT Client

Command to Publish message to enable Led Tracking on the Router

```
ubuntu@ubuntu-Attisoft:~$ mosquittpub -h 192.168.1.109 -t "command/action/ATTISOFT/0000116f905c/request" -m '{"Id":13,"ActionCommands":[{"Id":69,"Type":2,"Data":{"State":1,"Delay_M":3}}]}'
```

Command to subscribe Led Tracking message

```
ubuntu@ubuntu-Attisoft:~$ sudo mosquitto_sub -h 192.168.1.109 -t "command/action/ATTISOFT/0000116f905c/request"
{"Id":13,"ActionCommands":[{"Id":69,"Type":2,"Data":{"State":1,"Delay_M":3}}]}
```

Reboot the MQTT Client: RailBox

MQTT request to reboot the MQTT Client

```
ubuntu@ubuntu-Attisoft:~$ mosquitto_pub -h 192.168.1.109 -t "command/action/ATTISOFT/0000116f905c/request" -m '{"Id":2,"ActionCommands":[{"Type":8}]}'
```

MQTT request to reset to factory the MQTT Client

```
ubuntu@ubuntu-Attisoft:~$ mosquitto_pub -h 192.168.20.23 -t "command/update/laptop/00000fa1f308/request" -m '{"Id":66,"UpdateCommands":[{"Id":6,"Type":9}]}'
```

WIFI Scan the MQTT Client: RailBox

MQTT request to WIFI scan on the MQTT Client

```
ubuntu@ubuntu-Attisoft:~$ mosquitto_pub -h 192.168.1.109 -t "command/action/ATTISOFT/0000116f905c/request" -m '{"Id":2,"ActionCommands":[{"Type":3}]}'
```

NOTE: It should be noted that, according to the MQTT protocol, topic names are case-sensitive. For instance, topic aTTIsOFT is not the same as topic ATTISOFT.