APRS over MQTT Platform

Project Specification

Document Number: TBD

Status: Preliminary

Table of Contents

List of Tables

List of Figures

References

[1] gnu.org, "General Public Licence," [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html. [Accessed 25th February 2018].

[2] Wikipedia, "Automatic Packet Reporting System," [Online]. Available: https://en.wikipedia.org/wiki/Automatic_Packet_Reporting_System. [Accessed 22 August 2024].

[3] Wikipedia, "Internet of Things," [Online]. Available: https://en.wikipedia.org/wiki/Internet_of_things. [Accessed 22 August 2024].

[4] MQTT, "MQTT: The Standard for IoT Messaging," [Online]. Available: https://mqtt.org/. [Accessed 22 August 2024].

[5] Wikipedia, "OASIS (organization)," [Online]. Available: https://en.wikipedia.org/wiki/OASIS_(organization). [Accessed 22 August 2024].

[6] Bosch Sensortech, "BME680 Low power gas, pressure, temperature and humidity sensor," [Online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme680-ds001.pdf. [Accessed 22 August 2024].

[7] ST Microelectronics, "STM32H723," [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32h723-733.html. [Accessed 11 August 2024].

Revision Status

Revision	Date	Description
0.1	August 23, 2024	Initial draft
0.2	August 24, 2024	Corrected errors and added examples from code libaray

Table 1 Revision status

Intellectual Property Notice

Disclaimer

This document is a preliminary release for a product still in development and may be subject to change in future revisions. The software contained herein may be subject to unpredictable behaviour without notice. You are advised to keep a can of RAID™ Ant, Roach and Program Bug killer handy. Spray liberally on the affected area when needed.

Introduction

The Automatic Packet Reporting System (APRS) was introduced in the late 1980s by Bob Bruninga, WB4APR [2], a senior researcher at the US Naval academy, and has since grown to a world-wide network that reports position information as well as weather conditions and telemetry. It has become a tool used by amateurs worldwide for data acquisition and monitoring.

APRS relies on amateur frequencies and a digital packet format using audio frequency shift keying (AFSK) at 1200 bits/second. Data can originate at a mobile or fixed station, and to facilitate reporting to the APRS database, two types of receiving stations have been deployed, those that simply repeat what they hear to extend communications range (digipeaters), and those that provide a bridge to the commercial internet for reporting purposes (I-Gates). Access to the database is limited to the deployment of these station types, and their coverage is not ubiquitous.

Similarly, in the commercial arena, more and more devices are utilizing a technology commonly referred to as the 'Internet of Things' (IOT) [3] which acts in a similar manner. This has let to the development of a new service to support both stationary and mobile stations, and the emergence of an internet protocol specifically designed for this purpose, known as the Message Queuing Telemetry Transport, or MQTT [4]. As this is supported by commercial ventures, the coverage is extensive.

The objective of this project is to marry these two technologies together and create a platform that can bridge data sent over MQTT and post it to APRS. This not only opens up a global coverage area but enables new devices to be able to source data and add new reporting applications.

Some of the applications discussed here include monitoring the air quality index, for both stationary and mobile stations, as well as extending coverage for portable repeater applications in remote locations and adding repeater telemetry.

Overview of MQTT

MQTT is a lightweight publish/subscribe messaging protocol designed for M2M (machine to machine) telemetry in low bandwidth environments. It was designed by Andy Stanford-Clark (IBM) and Arlen Nipper in 1999 for connecting Oil Pipeline telemetry systems over satellite. Although it started as a proprietary protocol it was released royalty free in 2010 and became an OASIS (the Organization for the Advancement of Structured Information Standards) [5] standard in 2014.

The architecture of an MQTT system consists of a single central "broker", which takes data from client devices, known as "publishers", and sends them to receiving clients known as "subscribers". All devices are considered clients to the broker, and there can be any number in a system, and any client can be both a publisher and subscriber. Figure 1 illustrates a typical MQTT system.

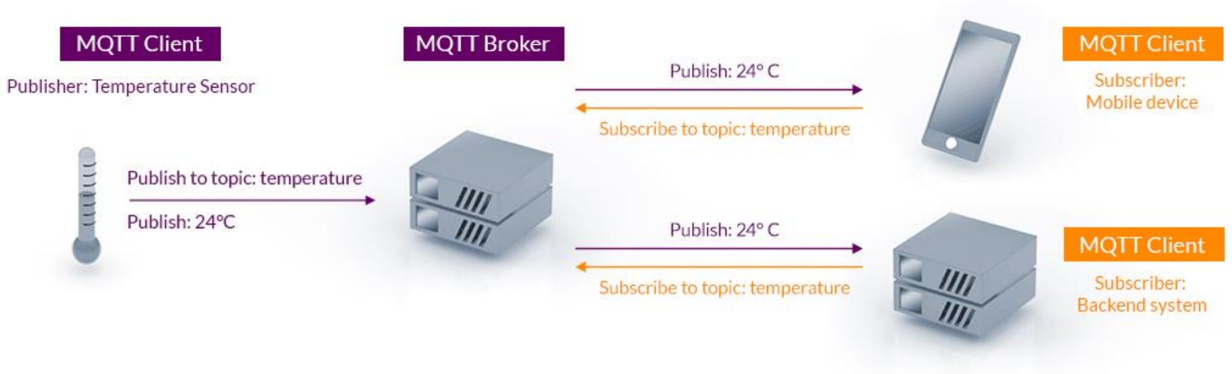

Figure 1 MQTT System architecture

In the example above, a temperature sensor publishes data to the broker, and two subscribers simultaneously receive the data from it.

There are currently two versions of MQTT in circulation V3 and V5. The older version is quite mature and well supported and has been deployed often enough to be problem free, hence it is the first to be adopted.

ADRCS Broker

The ADRCS society has installed an MQTT broker on its servers, using a popular open source offering from the eclipse foundation known as "Mosquitto". This is a self-contained software system written in the Java language.

The broker can be accessed internally on the AREDN mesh network, or externally over the commercial internet.

Access method	URL	Protocol	Port
Internal	va6edn-server.local.mesh	TCP	1883
External	aprs.adrcs.org	TCP	7000

Table 2 ADRCS broker access

Platform Components

Figure 2 illustrates the components of the APRS over MQTT platform.

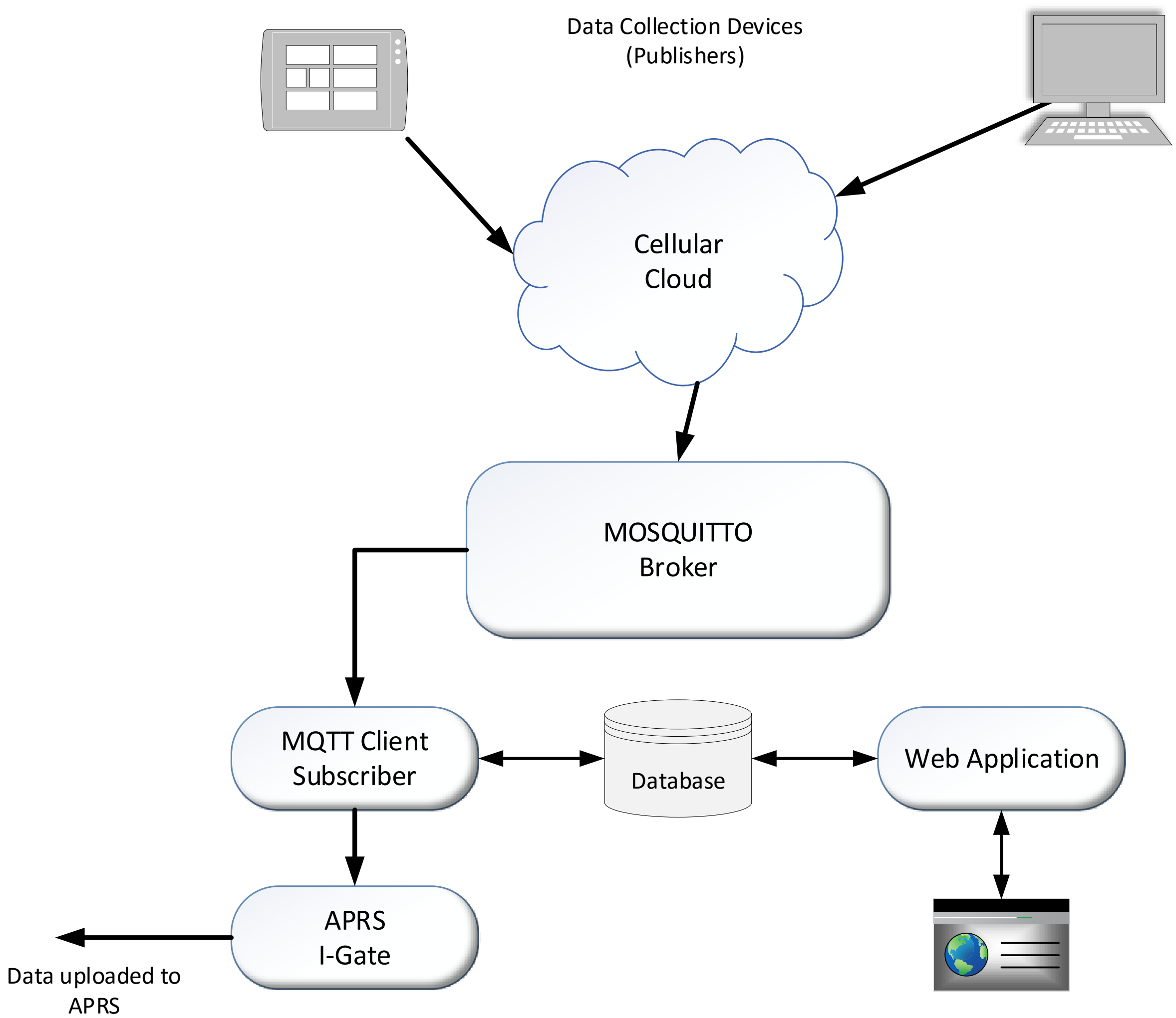

Figure 2 Platform Components

As previously discussed, the Mosquitto broker is the central component of the platform. The remainder consists of three components:

1. MQTT Client. A client to subscribe to data published by the collection devices. This parses the message and posts the data to a database, adds any required fields for APRS posting.
2. APRS I-Gate. This module takes the data from the database and sends it to APRS.
3. A web application provides a view into the database and provides methods to enter additional data required for posting to APRS.

MQTT Publishing Applications

There are three applications that are being considered for the platform:

1. Air Quality Index monitoring. An application to monitor the Air Quality Index from a stationary or mobile platform.
2. Repeater Cellular I-Gate. An addition to a repeater system to forward received APRS messages using the cellular network.
3. Telemetry. Telemetry received from a repeater system or other device that can be posted.

AQI Monitoring

The AQI monitoring application adds the capability to post an index to APRS from a fixed or mobile station using a combination of Temperature, Pressure, Humidity and the amount of Volatile Organic Compounds in the air. This data is extremely useful during wildfire seasons and after disasters such as oil spills from transportation systems.

The sensor that gathers the data is a BME680 by Bosch SensorTech [6], which is contained in a single surface mount component. There are two hardware systems that are being considered for this project:

1. Arduino UNO R4 Wi-Fi hardware with external sensor. This module is available of the shelf for fixed station use and contains an ARM® Cortex®-M4 processor, coupled with an Expressif ESP32 WiFi module. A library is available from Bosch to calculate the AQI, and the ESP 32 can send data using the MQTT protocol. This device is fixed as it requires an access point to connect to the internet and the broker.
2. Nordic Semiconductor 'Thingy91' module. This is a factory-built module that contains an NRF9160 cellular IOT processor that can connect to an LTE or NB-IOT network. The 9160 is a single chip that has two 64 MHz Arm® Cortex®-M33 CPU processors, one dedicated to the modem application, and the second for a user application. It also has an on-chip receiver for the Global Navigation Satellite System (GNSS), which can determine position information from the LTE network as well as satellites. The device also has a BME680 sensor, and a battery pack for portable use. Figure 3 illustrates the device.

Figure 3 Nordic Semiconductor Thingy91 prototyping platform

Repeater Cellular I-Gate

This application adds a local I-Gate capability to a repeater with the addition of the same prototyping platform as in the AQI project but acts as a 'store and forward' device for APRS messages. Figure 4 is a block diagram of the application.

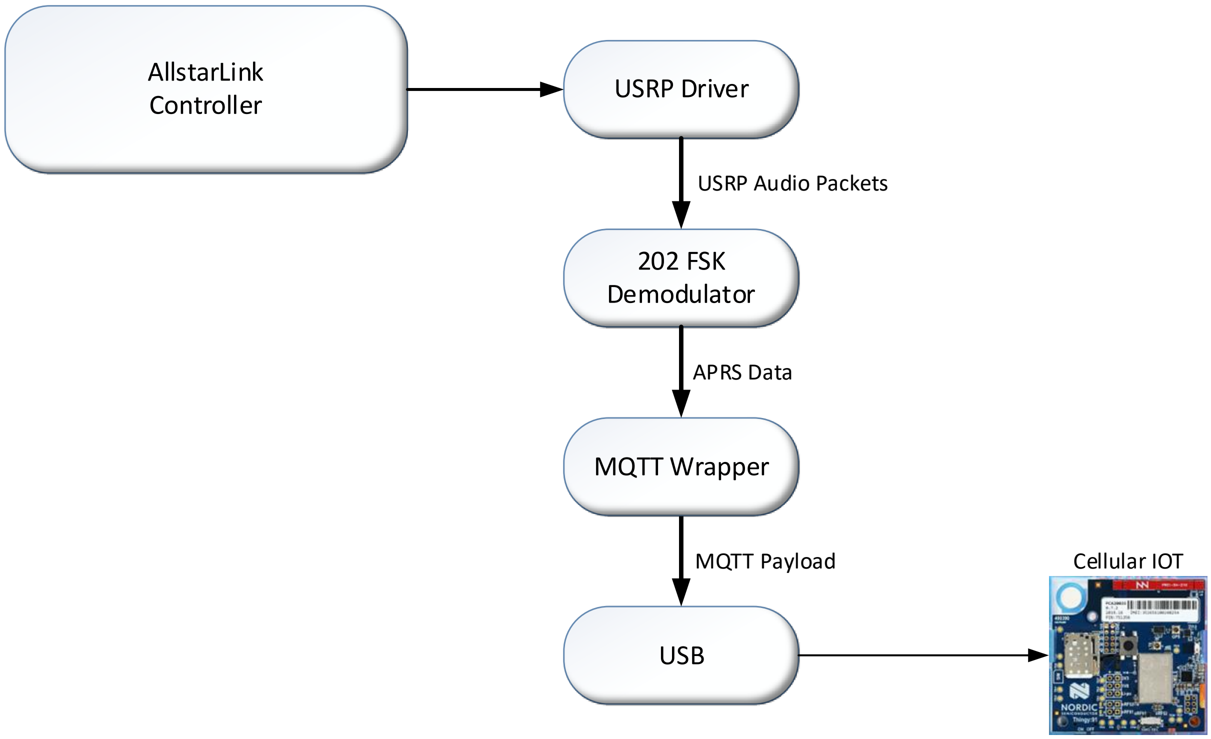

Figure 4 Repeater cellular I-Gate block diagram

Audio packets from a repeater control system such as AllstarLink, are capable of generating audio packets using the USRP (Universal Software Radio Peripheral) protocol. This contains up to 20ms of speech which is uncompressed and in a 16-bit linear PCM format. Several drivers are available for this protocol.

A soft modem receives the USRP packets over a UDP connection, and demodulates the APRS data from it, extracting both routing information as well as the packet payload. This is then passed onto a wrapper that condenses it into an MQTT payload. This is then passed on to the Cellular IOT platform using a USB UAR/T for transmission.

The software can coexist with the controller on the same hardware, such as a Raspberry PI, or any other platform that supports an ethernet and USB connection. When on the same platform, the loopback IP address (127.0.0.1) is utilized for USRP so the packets are not transmitted.

Telemetry

This application connects a PUTSI™ telemetry encoder to the cellular network, using a local WiFi hotspot. As in the I-Gate application, packets at the repeater level are encoded using the USRP protocol. Figure 5 illustrates the repeater end of the application.

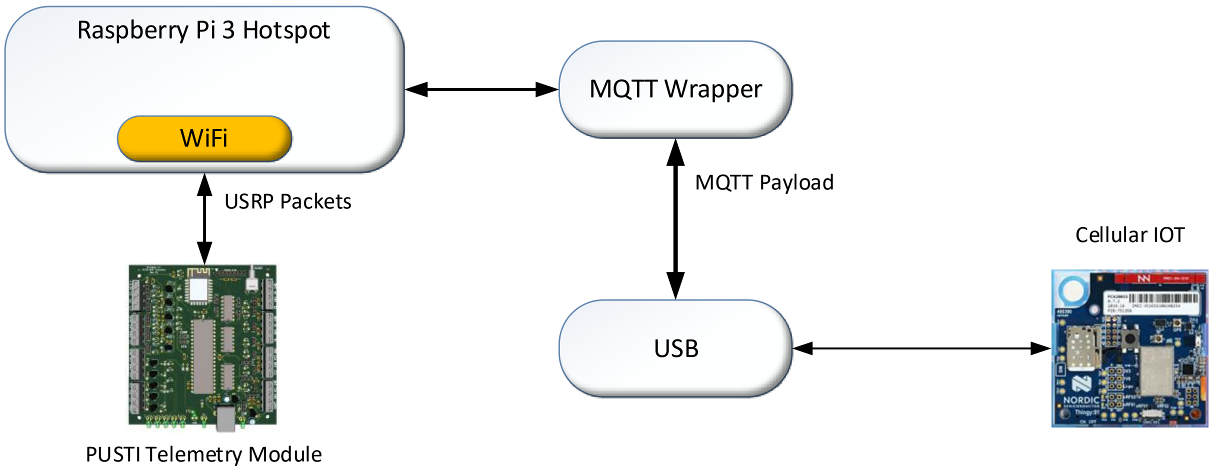

Figure 5 Repeater Telemetry Application

In this configuration a local hotspot provides a link to a PUTSI™ telemetry encoder, which has up to 8 analog inputs, as well as 5 dedicated digital outputs and inputs. Data is sent over the WiFi connection utilizing the USRP protocol. The cellular IOT device has the ability to publish telemetry data and also subscribe to commands to turn local I/O devices on or off.

MQTT Message Formats

All MQTT messages contain two fields of interest:

1. Topic. A field that contains a main topic and subtopics separated by slashes.
2. Payload. The data portion of the message. All payloads are sent in JSON format.

Topics

Table 3 lists the fields in the message topics:

Field Number	Contents
1	Constant “mqtt_aprs”. Identifies the message to be sent to the MQTT client
2	Application name. Specific to the application
3	Originating device. A short name for the originating device.

Table 3 MQTT Topics

Required Payload fields

All payloads contain common fields and those specific to the application. Table 4 lists the required fields for each message, and Table 5 lists the fields in the position information

Field Name	Type	Contents
Device ID	String	An identifier that uniquely identifies the device. Can be an IMEI or callsign.
Position	Array	An array containing position information, some fields are required
Battery Level	Number	An basic telemetry unit to identify the system power status.
Epoch Time	Number	A timestamp to uniquely identify the message. Can be TOD or uptime.

Table 4 Payload Common fields

Field Name	Type	Req’d	Contents
Latitude	Double	Yes	Latitude as a fraction, negative for southern hemisphere.
Longitude	Double		Longitude as a fraction, negative for western hemisphere.
Altitude	Number		Altitude above sea level in metres.
Speed	Number	Opt	Territorial or airspeed in kilometres/second.
Heading	Double		Compass heading for direction of travel if speed is non-zero
Valid	Boolean	Yes	Indicates position fix is valid when true.

Table 5 Position Information JSON array

Figure 6 illustrates an example of a message header in JSON format.

```
{
      "Device ID":  "MyDeviceID",
       "Position":  [{
            "Latitude":     51.08,
            "Longitude":    -114.1,
            "Altitude":     1000,
            "Speed":        0,
            "Heading":      264,
            "Valid":        false
         }],
         "Battery Level":  95,
         "Epoch Time":     1722356996379,
<<Application specific data goes in here>>
}
```
Figure 6 Header fields example

Specific application add additional data in the form of arrays uniquely identified by the information source.

AQI Application

Table 6 lists the fields required for the IAQ application, and "BME680":   [{

```
      "Temperature":  25,
       "Pressure":     1012,
       "Humidity":     10,
       "IAQ":  50
}]
```

Figure 7 shows an example.

Field	Type	Contents
First Sub Topic	String	The constant "AQI"
Second Sub Topic	String	An identifier for the device, ie nrf9160, ArduinoR4, etc.
BME680	Array	An array containing the BME680 data, all fields are required
Temperature	Double	Temperature in degrees C
Pressure	Double	Atmospheric pressure in KPa
Humidity	Double	Humidity in percent
IAQ	Number	IAQ in the range of 0-350

Table 6 Fields specific to the AQI application

```
"BME680":     [{
      "Temperature":  25,
       "Pressure":     1012,
       "Humidity":     10,
       "IAQ":  50
}]
```
Figure 7 IAQ application fields example

I-Gate application

Table 7 lists the fields for the I-Gate application, and "UIFrame": [{
```
      "Source":         "ve6nhm-10",
     "Destination":   "ve6nhm-2",
      "Repeaters":     "ve6nhm-10",
      "Frame":         "542356120013275ADE6F"
  }]
```

Figure 8 illustrates an example.

Field	Type	Contents
First Sub Topic	String	The constant “RPT”
Second Sub Topic	String	An identifier for the device, ie nrf9160, ArduinoR4, etc.
UIFrame	Array	An array containing the APRS UI frame
Source	String	Source station callsign and SSID as an ASCII string
Destination	String	Destination station callsign and SSID as an ASCII string
Repeaters	String	Repeater callsigns and SSIDs separated by commas as ASCII string
Frame	String	UI frame encoded as 2-byte HEX characters in ASCII

Table 7 Repeater I-Gate application fields

```
  "UIFrame":  [{
      "Source":         "ve6nhm-10",
     "Destination":   "ve6nhm-2",
      "Repeaters":     "ve6nhm-10",
      "Frame":         "542356120013275ADE6F"
  }]
```

Figure 8 I-Gate application example

Telemetry Application

Table 8 illustrates the fields in the telemetry application. When publishing from the endoder, all fields are required, but in the subscription (upload) direction, the ADC and input fields are omitted.

Field	Type	Pub/Subscribe	Contents
First Sub Topic	String	Both	The constant “Telemetry”
Second Sub Topic	String		Callsign of the repeater system
Radio_ID	Number		A unique radio identifier
Num_ADC	Number	Publish Only	Number of A/D input channels
ADCdata	Array		An array containing all the ADC values as numbers
Num_INP	Number		Number of digital inputs
INPData	Array		An array containing the inputs as Boolean values
Num_OUT	Number	Both	Number of digital outputs
OUTData	Array		An array containing the outputs as Boolean values

Table 8 Telemetry application fields


```
"Telemetry":    [{
     "RadioID":      302092,
     "Num_ADC":      3,
     "ADC_Data":     [{
           "0":    813,
           "1":    0,
           "2":    23
           }],
      "Num_INP":      2,
      "INP_Data":     [{
           "0":    false,
           "1":    true
           }],
      "Num_OUT":      2,
      "OUT_Data":     [{
           "0":    false,
           "1":    false
       }]
}]
```


Figure 9 Telemetry example