Grenze International Journal of Engineering and Technology, Jan Issue

A Method of Real-Time Datalink Support for Differential Navic Measurements

Ramesh Raju[1] and Dr. Raju Garudachar[2]
[1]PhD Scholar, Aerospace Engineering, School of Engineering & Technology, Bengaluru - 562112, India
Email: srameshraju74@gmail.com
[2]Adj. Professor, Electronics and Communication Engineering, School of Engineering & Technology, Bengaluru - 562112, India
Email: raju.garuda@yahoo.co.in

***Abstract*—Standalone NavIC receivers experience position errors typically < 20 m per specification. For applications that need meter-level or centimeter-level accuracies, this standalone NavIC receiver accuracy is not adequate. The Differential NavIC technique provides better position accuracy for such applications. To improve the rover station position accuracy, the Differential NavIC uses a reference station by providing measurements or correction data. The correction data typically broadcast over a transmission link and received by the rover station. The transmission link can be a radio transmission (LF, MF, HF, or UHF) or a mobile communication network using different communication protocols. In this paper, to transmit the Differential NavIC correction data from the reference station to the rover station, the use of a machine-to-machine (M2M) protocol called Message Queue Telemetry Transport (MQTT) is being explored.**

***Index Terms*— Message Queue Telemetry Transport (MQTT), Differential NavIC, Position, accuracy, Navigation, Internet of things (IoT), Push Protocol, MDTT Publish, MATT Subscribe.**

I. INTRODUCTION

Indian Regional Navigation Satellite System (IRNSS), operating with its name NavIC (Navigation with Indian Constellation), is a new technological implementation by the Indian Space Research Organisation (ISRO) like Global Positioning System (GPS) for helping the Indian Subcontinent and its neighbors on navigation and timing application. It consists of a constellation of seven satellites, with two additional satellites on the ground as stand-by. It has four geosynchronous satellites and three geostationary satellites. The purpose of this system is to have accurate real-time positioning and timing services to users in India as well as the region extending up to 1500 km from its boundary, which is its primary service area [1]. As per specification, the stand-alone NavIC can provide an accuracy of 20 m in its service area.

There are applications such as aerial & maritime navigation, defense/military applications, and few others demand high accuracy, integrity, availability, and continuity beyond the design specification of NavIC. Augmentation methods help to improve the system's attributes through integrating external information into the calculation process. One such augmentation method is Differential NavIC (D-NavIC).

D-NavIC is a method to improve the positioning and timing performance of NavIC using one or more Reference

Grenze ID: 01.GIJET.8.1.510

stations at known locations, each equipped with at least one NavIC receiver. Rover stations are the end-users of the D-NavIC, also equipped with at least one NavIC receiver. The Reference station calculates the error in the pseudo-range for each satellite using its known location and transmits it to the Rover through a real-time datalink. Using these error corrections, Rover improves its pseudo-range measurements and the satellite-provided clock and ephemeris data, integrity data, etc. Thereby Rover achieves better accuracy in position and time.

The data transmission can be over any communication channel, either via radio transmission (LF, MF, HF, UHF) or via a mobile communication network, using different communication protocols. The introduction and deployment of wireless Internet capability have opened up the potential of disseminating these data over the Internet [2].

The typical problems with wireless cellular Internet are:

- Vulnerable to various connectivity issues such as service outages, cellular signal blockages
- Limited data bandwidth, and
- The two-way communication may experience high latency and even loss of data communication

Message Queue Telemetry Transport (MQTT), a machine-to-machine (M2M) based protocol, establishes a reliable and low-latency two-way communication protocol [2]. This paper explores the usage of the MQTT protocol to transmit the error data between the Reference station to the Rover station.

II. MQTT PROTOCOL

MQTT is an open OASIS and ISO standard (ISO/IEC 20922) lightweight, a publish-subscribe network protocol that transports messages between devices. The protocol usually runs over TCP/IP; however, any network protocol that provides ordered, lossless, bi-directional connections can support MQTT [3].

A. Concepts

The MQTT protocol includes the following highlights [source: Ref. [4]]:

- Open and royalty-free for easy adoption: MQTT is an open-source, easy to adopt and adapt to the wide variety of devices, platforms, and operating systems that are used as end systems of the network.
- A publish/subscribe messaging model that facilitates one-to-many distribution: Sending applications or devices do not need to know anything about the receiver, not even its address.
- Ideal for constrained networks (low bandwidth, high latency, data limits, and fragile connections): MQTT message headers are kept as small as possible. The fixed header is just two bytes, and its on-demand, push-style message distribution keeps network utilization low.
- Multiple service levels allow flexibility in handling different types of messages: Developers can designate that message will be delivered at most once, at least once, or exactly once.
- Designed specifically for remote devices with little memory or processing power: Minimal headers, a small client footprint, and limited reliance on libraries make MQTT ideal for constrained devices.
- Easy to use and implement with a simple set of command messages: Many applications of MQTT can be accomplished using CONNECT, PUBLISH, SUBSCRIBE and DISCONNECT.
- Built-in support for loss of contact between client and server: The server is informed when a client connection breaks abnormally, allowing the message to be re-sent or preserved for later delivery.

B. Architecture

There are two main components in MQTT architecture as shown in Figure 1.

C. Client [4]

A program or device that uses MQTT. A Client:

- Open the Network Connection to the Server
- Publish the Application Messages that other Clients might be interested in.
- Subscribe to request Application Messages that it is interested in receiving.
- Unsubscribe to remove a request for Application Messages

Close the Network Connection to the Server

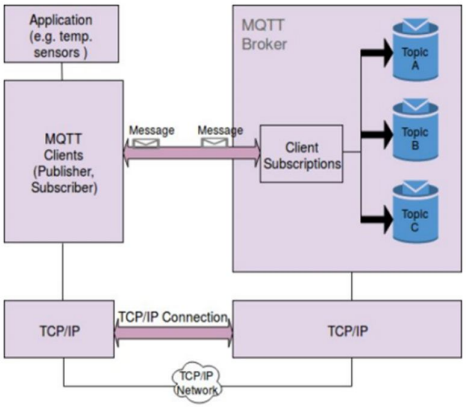

Figure 1. MQTT Architecture (source: Ref. [5])

D. Server or Brokert [4]

A program or device that acts as a medium between Clients who publish Application Messages and Clients who have made Subscriptions. A Server or Broker:

- Accept the Network Connections from Clients.
- Accept the Application Messages published by Clients.
- Process the Subscribe and Unsubscribe requests from Clients.
- Forward the Application Messages that match Client Subscriptions.
- Close the Network Connection from the Client.

III. MQTT D-NAVIC DATALINK APPLICATION

The MQTT-based D-NavIC Datalink Application system diagram is shown in Figure 2 transmits the Differential NavIC Error data from Reference Station to Rover Station.

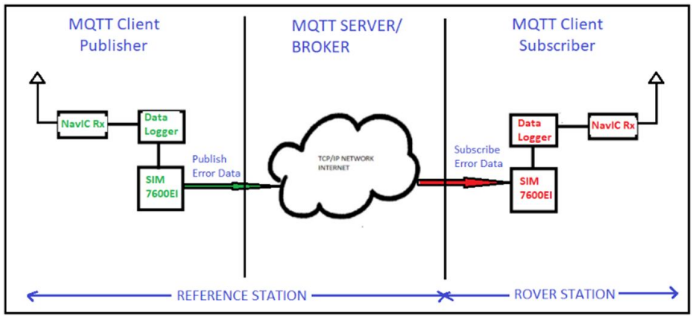

Figure 2. System Diagram: MQTT D-NavIC Datalink application

A. System Description

The system consists:

- Reference Station consists of NavIC Receiver and Datalogger installed at known Surveyed Marker Position.
- Rover Station consists of NavIC Receiver and Datalogger installed in a moving platform.
- Both Reference and Rover stations includes SIM7600EI 4G / 3G / 2G / GSM / GPRS / GPS UART Modem, a high-quality commercial-grade product from rhydoLABZ. The SIM7600E Modem has an internal TCP/IP stack to enable you to connect with the internet via 4G/GPRS [6].
- Eclipse Mosquitto [7], an open-source message broker that implements the MQTT protocol, is installed in Reference Station.
- Algorithms that fetch the Error data from Reference station and publish to the MQTT broker; subscribe the error data from MQTT broker and applies at Rover station. These algorithms are programmed using Python, which is an open-source language.

- Availability of Internet Service both at Reference and Rover Stations

B. System State Diagram

Experiments carried out with both Reference and Rover stations as MQTT Clients. Reference station with the algorithm that publishes the Error data each second to the MQTT Broker module. Rover establishes the connection to MQTT Broker and Subscribes the Error data message. MQTT Broker publishes the error data to the Rover as and when the data available. The system working model shown in Figure 3.

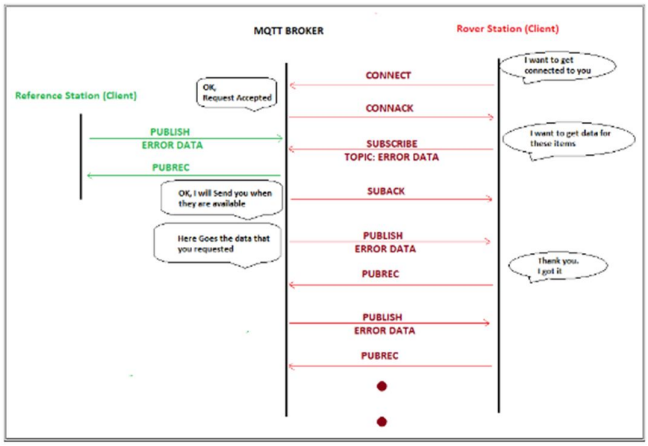

Figure 3. System Working Model: MQTT D-NavIC Datalink application [5]

C. Algorithms

Publish algorithm that fetch the error data from Reference station and publishes the messages to the MQTT Broker. The below figure 4, the code used a dummy 150-character size message to transmit from reference station to the Rover station every second.

```
# Modules to be imported for programming

import time
from time import sleep
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc): # checking for connection to broker
    print("Connected flags ",str(flags),"result code ",str(rc))

def on_publish(client, userdata, message): # checking the message is published
    print("message published " , message)# ,str(message.payload.decode("utf-8")))

MQTT_SERVER = "172.20.10.5"  #Write Broker IP Address (Should be Public IP when implemented on real time)
MQTT_PATH= "Data_from_base_station" #Topic Name
Data_Base_station = "awidfbwaijfbwiajfbswifvbairfgbvsjadkv14345289rsdsjdfnkANcfaufhajdkfbwefliusadfljasdbfiuwefjdsabfiuweabjabcoiduFYV8hfbou228745t924875t98dndasfbkwufbjew"
# 150 Character String

client = mqtt.Client() # object of client
client.username_pw_set(username="IOT_IRN", password="IOT0987") # Auntentication
client.on_connect = on_connect # call back on connection
client.on_publish = on_publish # call back on published message
client.connect(MQTT_SERVER, 1883, 60) # broker address
while True:
    client.publish(MQTT_PATH, Data_Base_station, qos=2)# Sending the data
    time.sleep(1) # Time Delay as desired
    client.loop()  # Continuous publish of Data

```

Figure 4. MQTT Publish Algorithm

Subscribe algorithm that subscribes the error data from MQTT Broker and provides the messages to the Rover station is in the below figure 5.

```
#Imporitng the modules for programming

import paho.mqtt.client as mqtt
import time
import datetime
from datetime import date

# connection feedback
def on_connect(client, userdata, flags, rc):  # The callback for when the client connects to the broker
       print("Connected with result code {0}".format(str(rc)))  # Print result of connection attempt

# Message received feedback
def on_message(client, userdata, msg):
    print(msg.topic + " " + str(msg.qos) + " " + str(msg.payload)) #msg with topic and Qos printed in client end

# Needed to include date and time for future
Current_date = datetime.datetime.now().strftime('%d-%b-%Y')  # Can be used in Suscribed date is needed
Current_Time = datetime.datetime.now().strftime('%H-%M-%S')  # Can be used in Suscribed time is needed

# subscription client
client = mqtt.Client() # object of client
client.username_pw_set(username="IOT_IRN",password="IOT0987") # Authentication
client.on_connect=on_connect # call back on connection
client.on_message=on_message # call back on received message
client.connect("172.20.10.5", 1883, 30) #Same broker address as the publish channel
while True:
    client.subscribe('Data_from_base_station',qos=2) # Subscribe the needed topic coming from the broker
    time.sleep(0.1) # delay if needed
    client.loop()# wait for the topic from the publisher
```

Figure 5. MQTT Subscriber Algorithm

D. Security

The Mosquitto MQTT broker is configurable to set a client authentication using a valid username and password before a connection is permitted.

E. Results

Experiments carried out and successfully transmitted the Error data from Reference Station to Rover Station. Figure 6 and Figure 7 show the messages (marked in the red circles) published by the Reference and messages received at the Rover. The message that published by Reference station is received by the Rover station every second.

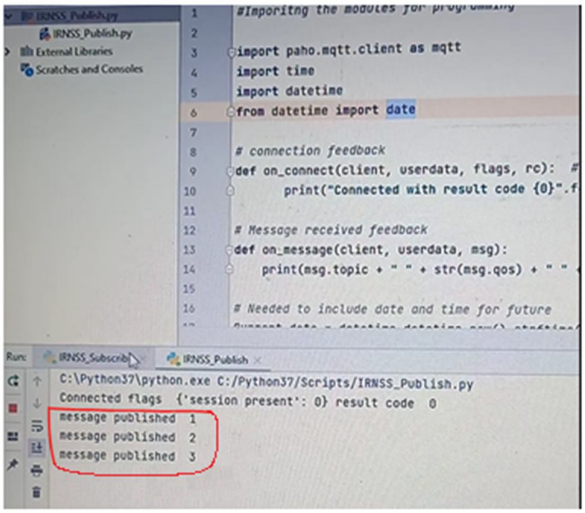

Figure 6: Screenshot of Publish messages by Reference Station

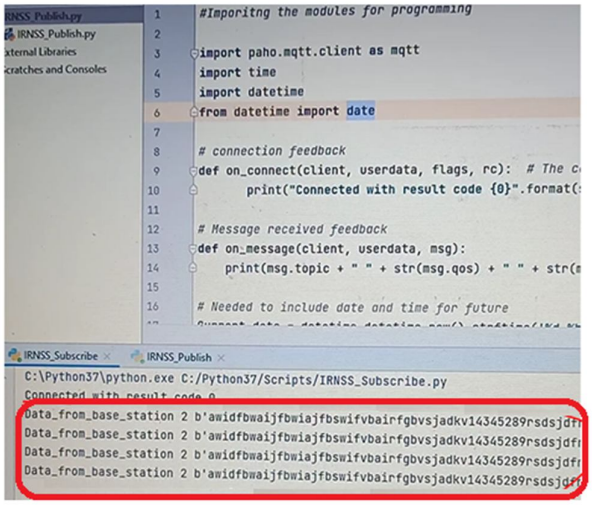

Figure 7: Screenshot of Received messages at Rover Station

IV. Conclusions

MQTT is the most popular communication protocol for wireless communication between various devices and systems. MQTT's simplicity and open-source code make this protocol suitable for constrained environments like IoT that have low power, limited computation capability and memory, and limited bandwidth. Our experiments prove it is suitable for the D-NavIC datalink applications that in turn enable the rover station to improve its position accuracy in real-time. It offers ranges more than 35 KM with no loss of data. MQTT usage nullifies problems like attenuation, LOS blockage, noise, co-channel interference, range issues, etc. As part of future work, the usage of the MQTT protocol can be further extended in the underlying implementation of NTRIP casters to improve latency and ensure data reliability in receiving NavIC data corrections via wireless connectivity.

Acknowledgment

The Authors are thankful to Space Applications Centre (SAC), Indian Space Research Organization (ISRO) in Ahmedabad for providing the NavIC receivers that are used for this experiment. The authors also acknowledge the staff of Electronics and Communication department in Jain University for providing extensive support to carry out experiments. We would like to thank my family members for the support and allowing me to spend quality time for this paper.

References

[1] Harsha Harde, Prof. M R Shahade, Diksha Badnore; Indian Regional Navigation Satellite Systems; 2015; e-ISSN: 2394-8299; p-ISSN: 2394-8280; Volume 1, Special Issue 1;
[2] Izwan Idris; Real-time vehicle monitoring and positioning using MQTT for Reliable Wireless connectivity; 2017; Research thesis, Faculty of Science and Engineering, Queensland University of Technology.
[3] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta; MATT-v5.0; 07 March 2019. OASIS Standard.
[4] Lampkin, V., Leong, W. T., Olivera, L., Rawat, S., Subrahmanyam, N., Xiang, R., ... & Locke, D; Building smarter planet solutions with mqtt and ibm websphere mq telemetry; IBM Redbooks; 2012.
[5] Luzuriaga, J. E., Cano, J. C., Calafate, C., Manzoni, P. A Survey on MQTT: A Protocol of internet of things (IOT); Conference Paper April 2017; Research Gate
[6] 4G / 3G / 2G / GSM / GPRS / GPS UART Modem; Product description and specification by rhydoLABZ; https://www.rhydolabz.com/wireless-gsm-gprs-c-130_185/sim7600ei-4g-3g-2g-gsm-gprs-gps-uart-modem-rhydolabz-p-2665.html
[7] R. A. Light; Mosquitto: server and client implementation of the MQTT protocol; The Journal of Open-Source Software, vol. 2, no. 13, May 2017, DOI: 10.21105/joss.00265