Cellular Internet of Things for Practitioners

Cellular Internet of Things for Practitioners

REZA VAHIDNIA AND F. JOHN DIAN

BRITISH COLUMBIA INSTITUTE OF TECHNOLOGY
VANCOUVER, CANADA

Sample APA-style citation (7th Edition):

Vahidnia, R., & Dian, F. J. (2021). *Cellular Internet of Things for Practitioners.* British Columbia Institute of Technology. https://pressbooks.bccampus.ca/cellulariot/

Ebook ISBN: 978-1-990132-01-8

Print ISBN: 978-1-990132-00-1

Publisher: BCIT

Publication date: January 30, 2021

This book was produced with Pressbooks (https://pressbooks.com) and rendered with Prince.

Contents

Dedicated to

our loving parents

About the Authors

F. John Dian is a faculty at the Department of Electrical and Computer Engineering at British Columbia Institute of Technology (BCIT) in Vancouver, Canada. He received his Ph.D. degree from Concordia University, Montreal, Canada in Electrical and Computer Engineering in 2004. Dr. Dian has more than 20 years of experience in design and implementation of telecommunication circuits and systems. Dr. Dian holds a certificate in business analytics from Harvard University, USA and co-chairs the center of excellence for analytics at BCIT. He has been recognized with several excellence in teaching and research awards. Dr. Dian is a senior member of the Institute of Electrical and Electronics Engineers (IEEE) and an active member of Association of Engineers and Geoscientists of British Columbia .

Reza Vahidnia is a faculty at the Department of Electrical and Computer Engineering at British Columbia Institute of Technology (BCIT) in Vancouver, Canada. He received his Ph.D. degree from Ontario Tech University, Oshawa, Canada in Electrical and Computer Engineering in 2014. He has served as a senior IoT analyst at Rogers Communications Inc., and as an IoT project manager at TELUS where he worked on several smart city and IoT initiatives. Dr. Vahidnia is a professional engineer (P.Eng) and an active member of the Association of Engineers and Geoscientists of British Columbia (APEGBC).

Preface

In today's Internet of Things (IoT) ecosystem, Low-power Wide-area (LPWA) communication technologies play a central role in connecting the smart objects to the Internet due to their low complexity, low power consumption, wide coverage and high capacity. Cellular IoT technologies such as LTE-M and NB-IoT bring significant value for LPWA use cases over non-cellular communications due to their ubiquitous connectivity, technology maturity, scalability, reliability and security.

This book, first describes the simplified architecture of an IoT network from the core functional perspective, then presents step-by-step procedures to establish a connection between the IoT device and platform. It practically shows how to connect a cellular IoT module to the Microsoft Azure IoT Hub using the LTE-M technology.

Four experiments are designed to practice different concepts required to exchange the data between the IoT device and the software backend. In Chapter 2, the required hardware and AT commands to configure the LTE-M module are described. This chapter will also show how to send text messages and enable the GPS module to acquire the positioning information.

The purpose of Chapter 3 is to configure the LTE-M module as a TCP/IP client in order to establish a client-server connection with a server and exchange data.

After becoming familiar with various IoT application protocols in Chapter 4, the MQTT function lab will show how to configure the LTE-M module to act as a MQTT client/subscriber and exchange data with a broker. The broker will be built on a computer using Node-RED programming tool to wire together hardware devices, and online services to establish a communication with the LTE-M module. The data published on the broker will be sent to the MQTT clients that have subscribed to the topic.

Finally, Microsoft Azure IoT Hub is introduced which is a cloud managed service acting as a central message hub for two-way secure communication between the IoT application and the IoT device it manages. In Chapter 6, you will learn how to

use the Node-RED flow to establish a connection between the Microsoft Azure IoT platform and the IoT device. Once the data is received by Microsoft Azure, different components can be utilized to visualize and analyze the data to create actionable insight.

Chapter 1: IoT Technology Stack

1.1 Introduction

The Internet of Things (IoT) has connected billions of sensors and devices and created many opportunities that led many businesses to initiate IoT marketing and development budgets. Affordable, smaller and more powerful hardware, ubiquitous connectivity, availability of big data tools and cloud-based services, and huge market awareness are some of the reasons why more and more IoT opportunities and use cases are occurring [1]. To bring these many use cases and opportunities to life and handle the huge amount of data generated by massive number of IoT devices and sensors, a solid infrastructure and architectural model is needed. From a network architectural standpoint, the simplified core IoT functional stack has four main layers as shown in Figure 1.1.
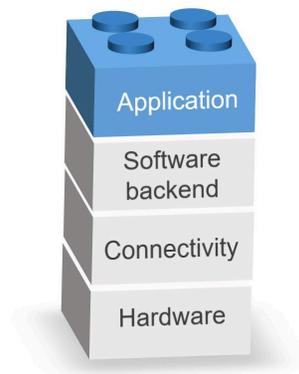
Figure1.1. Core IoT functional stack

Note that security is an essential element of each building block of this IoT technology stack.

A. Hardware

Hardware is where the data is collected and includes the smart objects (Things) with built-in sensors to measure physical data, actuators to perform tasks, low cost microprocessor, communication device to receive instructions, send or route data, and a power source (battery, mains, solar, etc.).

The characteristics and type of the smart objects in the hardware layer and their requirements can determine the technologies and protocols of the upper layers in the IoT functional stack. Battery-powered smart objects can have high mobility and require wireless technologies for their communications. However, their transmission range or reporting frequency could be limited. On the other hand, power-connected things are typically static and they are allowed to use technologies and protocols with higher power consumption that can provide longer transmission range and report their data more frequently. The amount and type of data collected by the smart objects and exchanged during the report cycles have an impact of the power consumption and consequently the choice of the connectivity technologies and protocols. For example, the electroencephalogram (EEG) signals collected by a smart helmet are richer compared to the data collected by a simple temperature sensor and hence, require more power and throughput to transmit the data.

Another important characteristic of a smart object is its report range or basically how far the data needs to be transmitted. For example, a vibration sensor installed on the train rails in a rural area may need to communicate with a cellular tower a few hundred meters away. Thus, for such a use case with a long report range, a low power communication technology is required to cover a wide area network.

The aforementioned hardware examples show that in order to design an IoT solution, a first step could be to examine the requirements of the "Thing" in terms of power consumption, report range, mobility, frequency of report and data transmission rate [2].

B. Connectivity

The connectivity or communications layer connects the smart object (hardware) to the network through an IoT access technology (e.g., WiFi, Bluetooth, etc.). Basically, the connectivity layer is responsible to transport the collected data from the sensors to the Internet through different access technologies with fundamentally different architectures, data transmission rates, report range, power consumption and security requirements. The majority of the existing IoT devices use one of the following forms of data transfer infrastructure shown in Figure1.2 to communicate with the software backend (cloud services) where the data and all connected devices are managed.

Figure1.2. IoT access technologies

Ad-hoc infrastructure

In this unlicensed short-range communication model, the "Thing" is connected to the Internet through a user-provided communications gateway or a vendor managed mesh network. In the user-provided architecture, the connectivity of smart object to the software backend (cloud or data storage infrastructure) is provided by technologies such as Bluetooth or WiFi that operate in the unlicensed spectrum. Although such technologies have high data transmission rates, security, provisioning and device management problems (e.g., different WiFi standards and security models, passwords change etc.) are major barriers to extend these

technologies into industrial and enterprise use cases. Moreover, the high power consumption (limited battery life cycle of the IoT devices and higher maintenance costs) and low reporting range of such ad-hoc networks could be other barriers to adopt these unlicensed spectrum technologies in a broader range of IoT use cases.

In vendor managed mesh networks, communication protocols such as ZigBee or Z-Wave are deployed to transmit the collected data into a gateway which then forwards the received data into the software backend usually via cellular networks (4G/LTE). In these conventional costly and power hungry solutions, different libraries need to be set up for each different type of simple or complex smart object to integrate them into the IoT platform [1].

Unlicensed spectrum technologies

LoRa and Sigfox are two low power, widely used and relatively cheap IoT technologies intended for battery operated smart objects with high report range (wide area networks). These unlicensed spectrum technologies target key IoT requirements such as secure bi-directional communications, mobility and localization services; however, their low data rate limits the capability to update the firmware and send complex commands. The maximum data rates for LoRa and Sigfox are 27kbps and 100bps, respectively.   Interference in high volume industrial applications is another issue in deploying these technologies. Moreover, the costly deployment of the technology and the need to set up a physical network infrastructure is another barrier in adoption of these alternative solutions.

Cellular Low-Power Wide-Area (LPWA)

One main important advantage of the cellular technologies over other unlicensed LPWA access technologies is their ubiquitous connectivity. The cellular networks are almost always available even in remote locations. Cellular IoT (CIoT) technologies are highly reliable, secure, and scalable which makes them suitable candidate technologies for massive IoT where a huge number of smart objects (up to 1 million devices per km^2) are connected to the Internet. Moreover, cellular

communications' maturity provides global ecosystem interoperability for CIoT technologies.

The 3rd Generation Partnership Project (3GPP) – a global technical body which develops technical specifications for mobile communication system – introduced a suite of two complementary CIoT technologies called enhanced Machine-Type Communications (eMTC or LTE-M), and Narrow band Internet of Things (NB-IoT) in its Release 13 [3].

LTE-M is a low-power wide-area (LPWA) air interface in the licensed spectrum that lets IoT devices connect with a data rate up to 1Mbps [4]. Thanks to two innovations of LTE Cat-M1 introduced by 3GPP Release 13, Power Saving Mode (PSM) and extended Discontinuous Reception (eDRx) longer battery lifecycles (up to a few years) are enabled and greater in-building range is supported as compared to standard cellular technologies such as 3G or LTE Cat 1. The power efficient LTE-M also supports voice functionality via VoLTE (used for applications requiring a level of human interaction), as well as full mobility and in-vehicle hand-over (used for asset tracking applications).

NB-IoT uses a small bandwidth for data transmission and reduces the bandwidth to 200 kHz (180 kHz plus guard-band) as compared to 1.4 MHz used in LTE-M [3]. Also, compared to LTE-M, NB-IoT consumes less power (longer battery life cycle) and provides even deeper coverage. The reduced complexity of NB-IoT makes it suitable for low data rate (10's of kbps), delay-tolerant use cases including sensors and meters.

C. Software backend

Software backend consists of cloud services that manage the network and the IoT devices. The integration of data and the interface to the 3rd party systems such as Enterprise Resource Planning (ERP) systems are provided by these cloud services. IoT platforms such as Microsoft Azure IoT Hub, Amazon Web Services (AWS) IoT, IBM Watson IoT are part of the central software backend. Event processing and action management, advanced analytics, privacy management, storage/database,

device management, and integration with external interfaces are some of the functionalities of the IoT platforms.

D. Application

The application layer visualizes the collected data from the sensors in real-time and integrates the business systems. This is how the raw collected data from the sensors is turned into value for businesses and companies. Moreover, the behavior of the smart things is controlled by the commands sent from the application layer to the smart objects.

Typically, there are two types of analytics in the application layer that generate two different intelligent views about the IoT system and the IoT network. First, data analytics which provides insight about the IoT system through analysing and processing the data collected by the sensors. Second, network analytics that evaluates the connectivity of the network to maintain the quality of service and prevent future network failures. The IoT network architect determines the depth of the network analytics depending on the use case. For use cases such as environmental monitoring where the sensors occasionally transmit their collected data without the need for an immediate action, a basic network analytics could be sufficient. On the other hand, in mission critical use cases such as connected vehicle, it is extremely impotent to have a detailed view of the network connectivity and performance.

Chapter 2: LTE-M Modem and AT Commands

2.1 Introduction

In the previous chapter, we briefly discussed different building blocks (layers) of an IoT solution from the core functional perspective. In this chapter and the next few chapters, we will discuss how to practically work with an LTE-M module to establish a CIoT connectivity scheme to transfer the data to the Internet. The high level architecture of the IoT project that we want to complete in this book is described in Figure2.1.
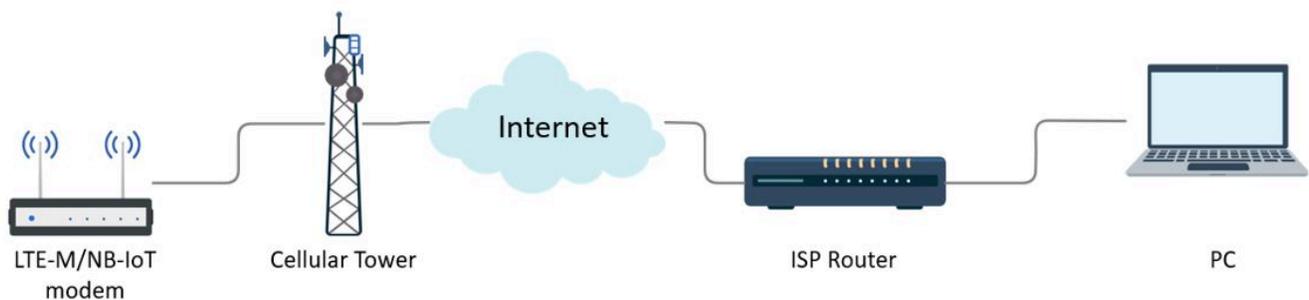
Figure2.1. High level architectural design of the IoT project

In this project, we will be using a Quectel BG96 LTE-M modem to exchange the data with the Internet through the cellular tower. The transmitted data is then stored and analyzed in the software backend (cloud platform) that gives full accessibility to the user to monitor, control and manage the device and network.

Following the step-by-step instructions in this chapter, you will become familiar with the TELUS IoT Starter Kit and Avnet Quectel BG96 LTE Cat-M1/NB-1 Shield. Also, you will be able to configure the Quectel BG96 modem using AT commands (instructions used to control a modem). Moreover, you will learn how to enable the GPS and acquire the location information using the AT commands.

In these lab series, you will learn how to create, prototype and develop an IoT product using the [TELUS LTE-M IoT Starter Kit](#)[1]. You will practice how to connect to the LTE-M network (The modem supports both LTE-M and NB-IoT. However, in this book, you will only work with LTE-M). Then, you will use Microsoft Azure cloud services to manage, store and analyze the data. The starter kit integrates three stacked boards as shown in Figure 2.2. A brief introduction of each board is given here. Note that in this book you will only work with the BG96 module (top board) in the standalone mode.
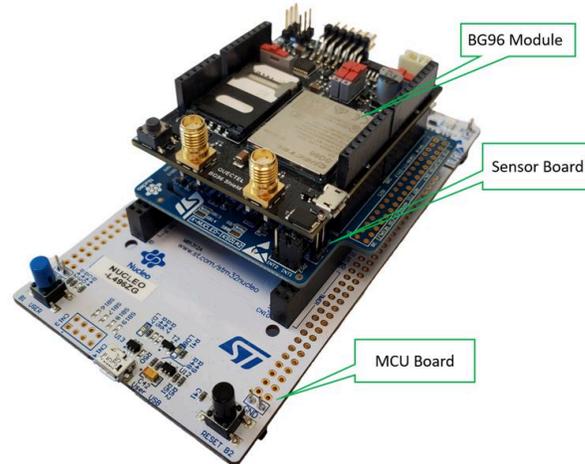
Figure2.2. TELUS IoT starter kit

2.2 Practical Tasks

A. BG96 Module

[BG96](#)[2] is a series of ultra low-power (approximately 10 μA in PSM mode) dual-mode LTE Cat-M1/Cat-NB1/EGPRS module manufactured by Quectel Wireless Solutions. The maximum uplink (UL) and downlink (DL) data rates for this module are 375kbps and 300kbps, respectively. This module is certified by the major mobile

---

1. http://cloudconnectkits.org/product/telus-lte-m-iot-starter-kit
2. https://www.quectel.com/product/bg96.htm

network operators in the world (TELUS, Bell and Rogers in Canada) and has wide range of IoT use cases such as asset tracking, smart cattle, etc.

The module includes the BG96 modem and is equipped with some additional features such as SIM holder, Arduino shield interface (connectors compatible with Arduino extension boards such as WiFi), LTE and GPS antenna SMA connectors, USB port and configuration switches.
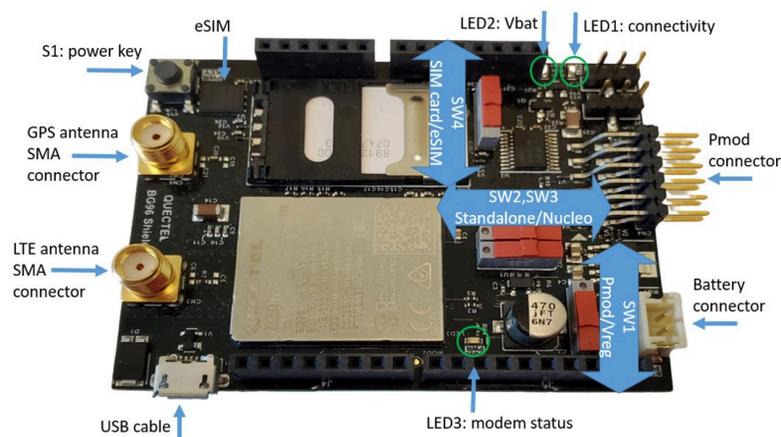
Figure2.3. Quectel BG96 module

B. Sensor Board

The [IKS01A2 Sensor Board](#)[3] contains the following sensors:

- [LSM6DSL](#)[4] 3D Accelerometer and 3D Gyro Sensor
- LSM303AGR[5] 3D Accelerometer and 3D Magnetometer
- LPS22HB[6] Barometric Pressure Sensor
- HTS221[7] Relative Humidity & Temperature Sensor

3. https://os.mbed.com/components/X-NUCLEO-IKS01A2/
4. https://www.st.com/en/mems-and-sensors/lsm6dsl.html
5. https://www.st.com/en/mems-and-sensors/lsm303agr.html
6. https://www.st.com/content/st_com/en/products/mems-and-sensors/pressure-sensors/lps22hb.html
7. https://www.st.com/content/st_com/en/products/mems-and-sensors/pressure-sensors/

The communication between this sensor board the MCU board (NUCLEO-L496ZG) is through I²C interface.

Figure2.4. Sensor board

C. MCU Board

Figure2.5 shows the NUCLEO-L496ZG Microcontroller Board[8] with Arduino connectors. You can consider connecting the BG96 module to this MCU board for future IoT products.
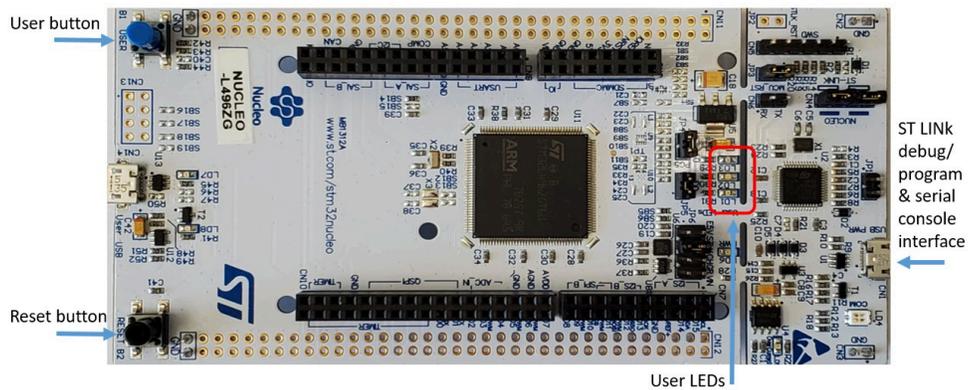
Figure2.5. STMicro NUCLEO-L496ZG MCU board

lps22hb.html

8. https://os.mbed.com/platforms/ST-Nucleo-L496ZG/

Note that in this book our focus is only on the BG96 module and we will not connect the sensor and the microcontroller boards to build the stack. If you are interested, you can build the full stack and try to send the AT commands to the BG96 module through the microcontroller board.

D. SIM card activation

In this sub-section, we discuss how to activate the SIM card for TELUS network operator. Different mobile network operators have different procedures to activate their SIM cards. If you already have an activated SIM card, skip this sub-section.

Before activating the TELUS SIM card, you need to have two codes handy:

- ICCID: Integrated Circuit Card Identifier (ICCID) is a 19-digit number located at the bottom of the SIM card.
- Starter Kit activation referral code. The referral code is provided for you at the back of the "Quick Starter Guide" page that came with TELUS LTE-M IoT Starter Kit[9] (e.g., **TELU8284**). If you do not have one, you can request a referral code at telus.m2m.com.

1. Go to telus.m2m.com -> Click "Enter Referral Code" -> Click "Order Kit" to create your account.
2. Create a username/password and enter your contact information and SIM card number (ICCID) and click continue. It may take a few days before TELUS can activate your SIM card and provide you with an Access Point Name (APN) through email. The provided TELUS APN could be "m2m-west.telus.iot" or "m2m-east.telus.iot" depending on your region in Canada. You will need this APN later when you want to connect to TELUS LTE-M network in the next chapters. An APN defines where a device will connect to the Internet. In order for a device to connect successfully to the Internet, the device application must use an APN that is allowed by the device's communication plan.
3. You can log in to TELUS Control Center[10] using your username/password to

9. http://cloudconnectkits.org/product/telus-lte-m-iot-starter-kit
10. https://telus.jasperwireless.com/provision/jsp/login.jsp

manage your device (view cycle-to-date usage, modify rate plan or SIM status), access your invoices, or modify your profile.
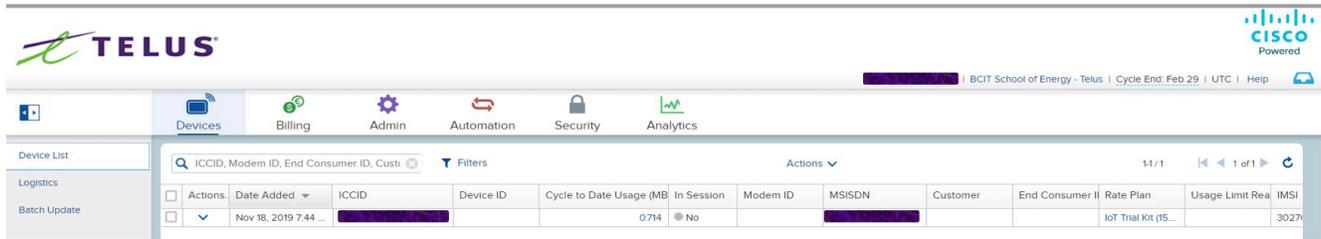
Figure2.6. Snapshot of TELUS Control Center

E. Standalone modem setup

In this part of the lab you will work with the BG96 board on its own and when it is not stacked to the MCU board.

1. You may detach the BG96 board from the Sensor Board and MCU Board. You can still use the BG96 board in standalone mode when it is stacked on the MCU board. However, make sure Switch S2 and S3 are positioned properly and the USB cable is connected to the USB port of the BG96 board and NOT that of the MCU.
2. Slide Switch S2,S3 towards the standalone position (to the left side as shown in Figure2.7)
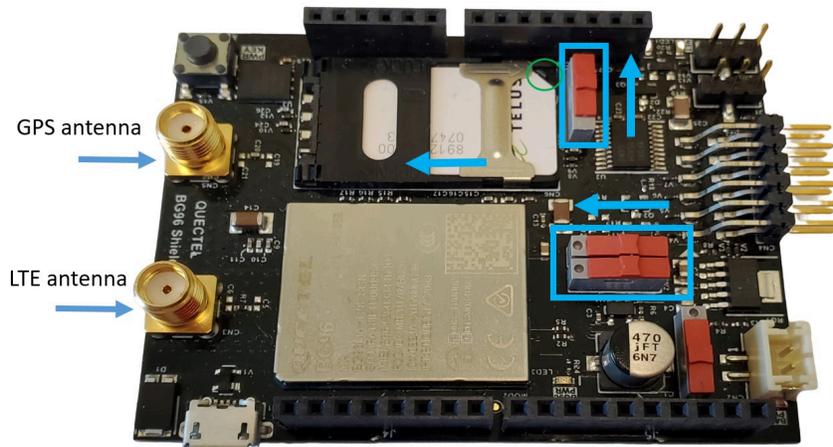
Figure2.7. Configuration of switches for standalone operation

3. Slide the metal retaining clip of the SIM holder, in direction of the SMA connectors (Left direction of Figure2.7), and then lift it up. Insert the activated SIM card with notched corner facing outwards. Press on the SIM holder and slide the metal retaining clip back in the direction of Pmod connector (Right direction of Figure2.7).
4. Set the sliding Switch SW4 to choose the SIM card.
5. Attach the LTE and GPS antennas to the SMA connectors.

GPS Antenna
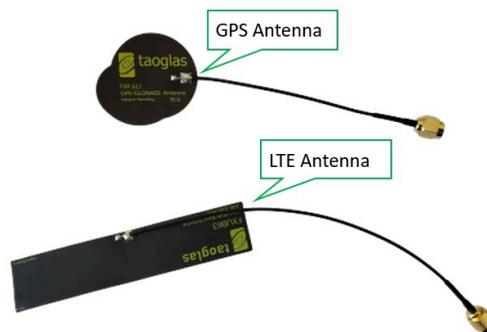
LTE Antenna

Figure2.8. GPS and LTE antennas

6. Connect your PC to the BG96 board's micro USB connector.
7. Press and hold Switch S1 (power key) for two seconds to wake up the modem. When the modem is awake, LED3 (modem status) should be off.

8. Download and install "Quectel_LTE_Windows_USB_Driver_V1.0[11]" or the latest version.
9. Go to "Device Manager" -> Ports (COM & LPT). You should see three Quectel USB ports:

   - Quectel USB AT port,
   - Quectel USB DM port,
   - Quectel USB NMEA port
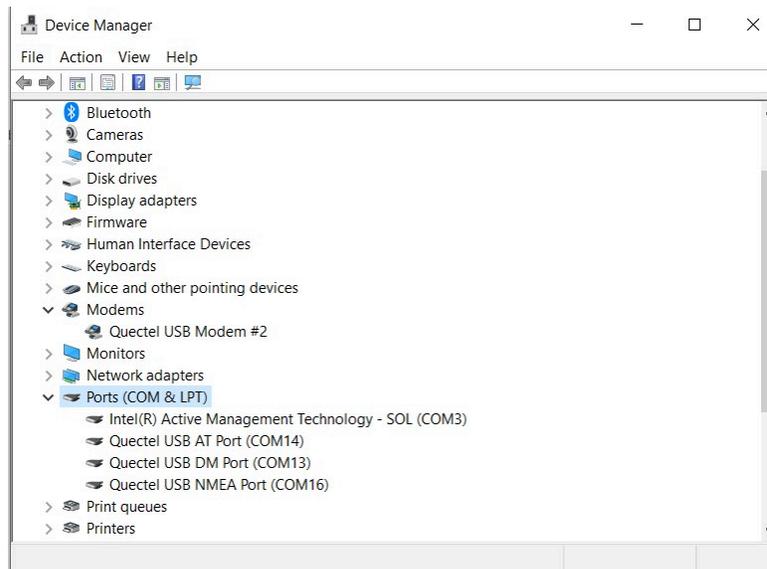
Figure2.9. Quectel USB ports

F. Quectel BG96 modem configuration using AT commands

AT commands (AT is the abbreviation of ATtention) are instructions used to control a modem and start with "AT". Quectel_BG96_AT_Commands_Manual[12] contains a full set of BG96 modem AT commands.

11. https://github.com/pengphei/quectel-ec20/tree/master/drv/
    Quectel_LTE_Windows_USB_Driver_V1.0
12. https://www.quectel.com/UploadImage/Downlad/
    Quectel_BG96_AT_Commands_Manual_V2.1.pdf

10. On your PC, open a terminal communications program such as **Tera Term** to emulate different types of computer terminals. If you do not have **Tera Term** installed on your PC, you can download the latest version at: https://osdn.net/projects/ttssh2/releases/

11. Configure the settings of the serial console application of **Tera Term** and create a new serial connection using Quectel USB AT Port.
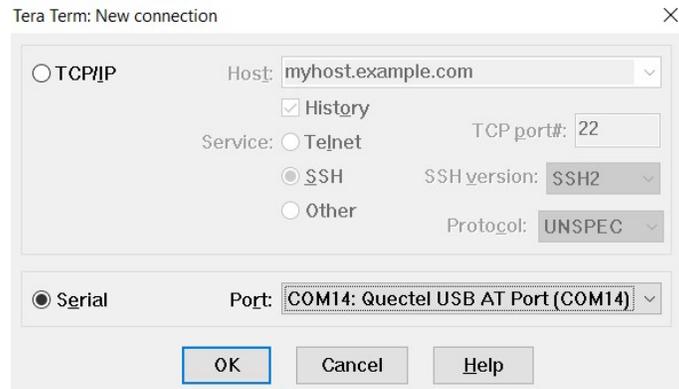
Figure2.10. Tera Term new connection

12. Go to "Setup" -> Click on "Serial port…" and set the Speed=115200, Data=8 bits, Parity=none, Stop bits=1 bit. Press "New Setting" to save these changes.
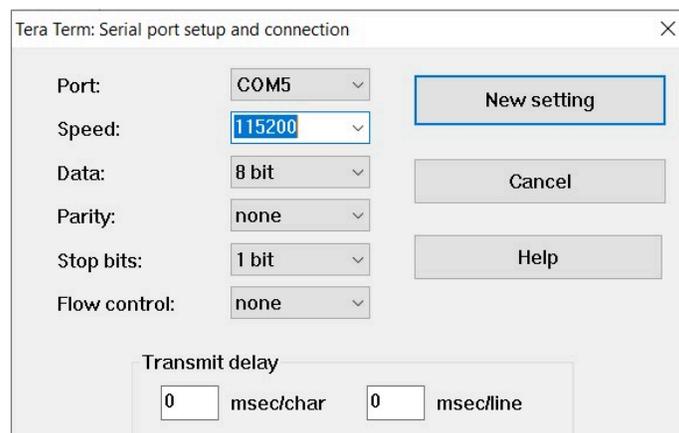
Figure2.11. Tera Term serial port setup

13. Go to "Setup" -> Click on "Terminal..." and make sure "Local Echo" is checked so that you can see the commands you will be typing.

14. Enter the following AT commands shown in **bold** letters. Below you can also see a brief description and the response of each AT command. The following configurations are based on TELUS as the current carrier. If your carrier requires a different configuration, modify the AT commands accordingly. For full description of AT commands, see Quectel_BG96_AT_Commands_Manual[13].

ATI *// Display Product Identification Information*

Quectel

BG96

Revision: BG96MAR02A07M1G

OK

AT GSN *// Request International Mobile Equipment Identity (IMEI). Note that IMEI is a unique 15-digit identification or serial number of the module.*

866425033446404

OK

AT CFUN=1 *// Set Phone Functionality as Full*

OK

13. https://www.quectel.com/UploadImage/Downlad/ Quectel_BG96_AT_Commands_Manual_V2.1.pdf

AT QCCID *// Show the SIM card's Integrated Circuit Card Identifier (ICCID) number*

QCCID: 89122301000074759513F

OK

AT COPS=0 *// Automatic Operator selection*

OK

AT QCFG="NWSCANSEQ",020301 *// Configure Radio Access Technologies (RAT) searching sequence*

02: LTE-M

03: NB-IoT

01: GSM

OK

AT QCFG="IOTOPMODE",0 *// Configure network category to be searched under LTE RAT.*

0: LTE Cat M1

1: LTE Cat NB1

2: LTE Cat M1 and Cat NB1

OK

AT QCFG="BAND",0,80A,80A *// Band configuration. Specify the frequency bands allowed to be searched of User Equipment (UE). The hex number represent the GSM, LTE-M and NB-IoT band values, respectively.*

0: not to change frequency band.

80A : LTE B2 LTE B4 LTE B12

OK

AT CSQ *// Signal quality report. Indicate the received signal strength indicator (RSSI) and the channel bit error rate (BER). For example the first received number (23) means the RSSI is -67 dBm and the second number (99) shows the channel bit error rate in percent.*

0: Less than or equal to -113dBm

1: -111dBm

2...30: -109... -53dBm

31: Greater than or equal to -51dBm

99: Not known

CSQ: 23,99

OK

AT COPS=? *// Operator Status.*

0: Unknown

1: Available

2: Current operator

3: Forbidden

As you see in the response from the module the status for TELUS is 2 which means it is the current operator.

COPS: (1,"Bell","Bell","302610",8),(1,"Rogers Wireless","ROGERS","302720",8),(1,"Rogers Wireless","ROGERS","302720",0),(2,"TELUS","TELUS","302220",8),,(0,1,2,3,4),(0,1,2)

OK

G. GPS test using AT commands

15. Enter the following simple AT commands to enable the GPS and acquire location information. For a complete set of Global Navigation Satellite System (GNSS) commands see BG96 GNSS AT Commands Manual[14].

AT QGPS=1   *// Turn on GNSS*

OK

AT QGPSLOC? *// Acquire positioning information*

QGPSLOC: 012102.0,4919.1986N,12305.5925W,3.2,-19.0,2,177.06,0.0,0.0,230220,08

OK

AT QGPSEND   *// Turn off GNSS*

OK

Note: If you are running the lab in a building, after you enter **AT QGPSLOC?**, you may get " CME ERROR: 516". This error means the modem has not received a GPS fix yet. You can wait for a few more minutes or move the GPS antenna near a window or outside. Note that a GPS fix is the locational information that the **GPS** system provides for a specific point.

16. In order to test the GNSS commands using NMEA USB port, download VisualGPS[15]. Note that NMEA is a standard data format that is supported by

---

14. https://www.quectel.com/UploadImage/Downlad/
Quectel_BG96_GNSS_AT_Commands_Manual_V1.1.pdf

GPS manufacturers. Run the VisualGPS application and go to "Connect to GPS" -> Click on "Connect using serial port" to graphically display the location.

17. Choose "Quectel USB NMEA Port", set the baud rate=115200 and press "Ok". As it is shown in Figure2.12, you will see the GPS data such as latitude, longitude, attitude, signal quality and etc.

Figure2.12. VisualGSP

H. SMS text messaging

If your carrier connectivity subscription includes SMS text messaging you will be able to send a text message using AT commands.

18. Enter the following commands:

15. http://www.visualgps.net/#visualgps-content

```
AT CMGF=1          // SMS message format is set as text

OK

AT CSCS="GSM"   // Character set is GSM

OK

AT CMGS="12369912020"  // Destination phone number

> Hello, this is a test.      // Enter your message and then press CTR-Z to send

CMGS: 5                        // Message reference upon successful delivery

OK
```

Chapter 3: TCP/IP AT Commands

3.1 Introduction

In this chapter, we would like to connect the BG96 board to the Internet to send and receive data. For this purpose, the BG96 board needs to establish a TCP/IP connection to a server. This can be done by using a software program such as SocketTest to enable the computer to act as a server. The configuration used in this lab is shown in Figure3.1.
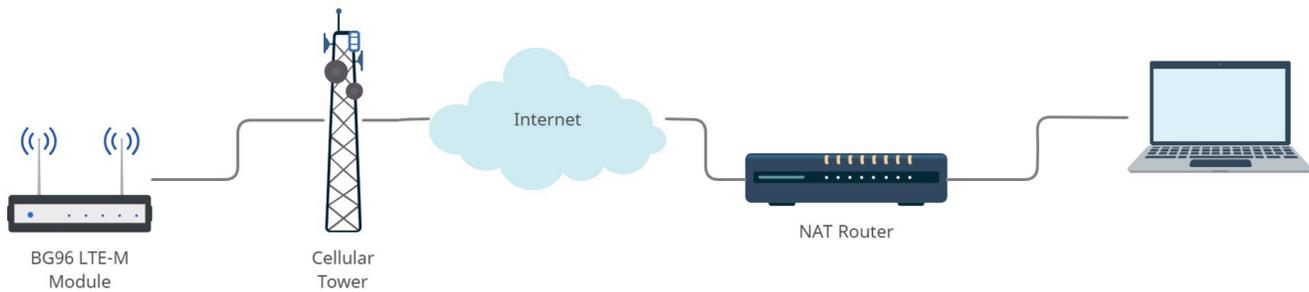
Figure3.1. BG96 connection to the Internet

For the BG96 board to communicate with the server installed on the computer, the BG96 board sends the data to the IP address of the computer. If the computer is installed behind a Network Address Translation (NAT) router like the ones in most homes and offices, the computer may use an internal address. For proper communication to take place, the NAT router needs to work well and proper port forwarding needs to be configured.

Quectel BG96 module can access to the Internet using the embedded TCP/IP stack and AT Commands. The module supports services for UDP client/server and TCP client/server. In this lab, the BG96 module will act as a TCP client and connects to a TCP server to exchange data.

3.2 Practical tasks

To start using TCP/IP AT Commands, first you need to configure the parameters of the context such as the APN, username, and password. Second, the Packet Data Protocol (PDP) context which is is the connection between the modem and the end address is activated. Third, you will start a socket service to exchange data. Finally the socket service is closed and PDP context is deactivated.

A. Configure your PC as TCP/IP server

In this part of the lab, you will set up a TCP/IP server on your computer to be able to communicate with the BG96 module.

1. Go to "Control panel"->"Windows Defender Firewall". Click on "Turn Windows Defender Firewall on or off" and turn off Windows Firewall or any third party firewall such Norton's or hardware firewall.
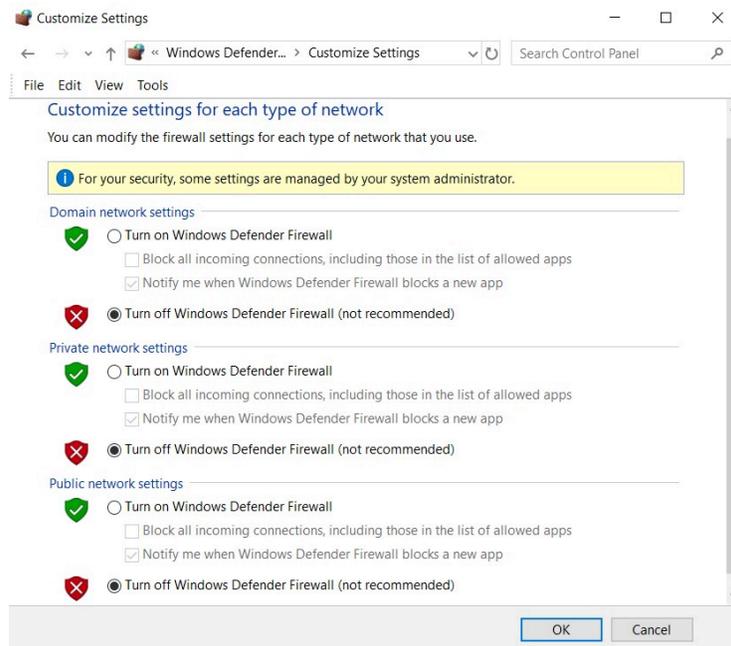
Figure3.2. Windows defender

2. To allow remote access to your PC, go to "Control Panel" -> "System"->"Remote settings". In the "Remote" tab select "Allow remote connections to this computer".
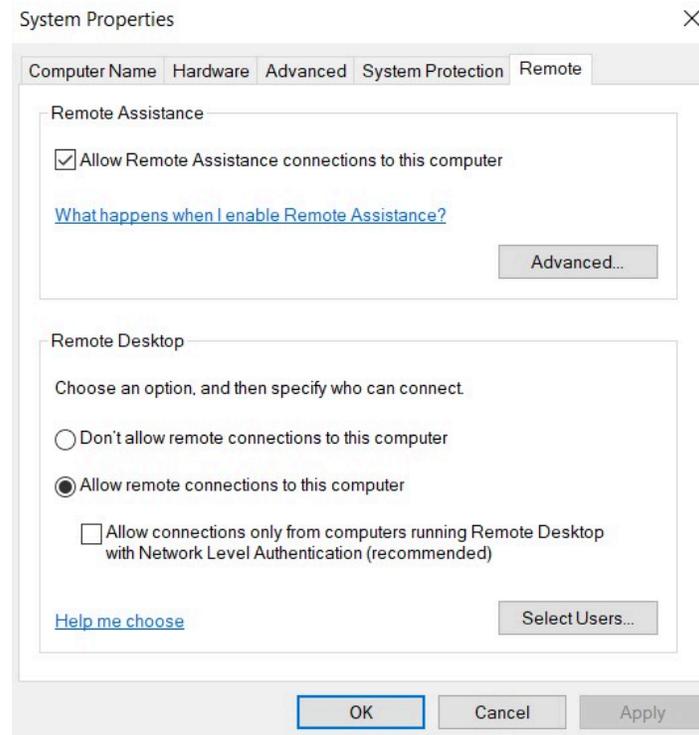
Figure3.3. Allow remote connections

3. To create a TCP/IP server, download and run SocketTest[1]. Click on "Server" tab and enter a number (e.g., 350) as your port number. Click "Start Listening".

1. http://sockettest.sourceforge.net/
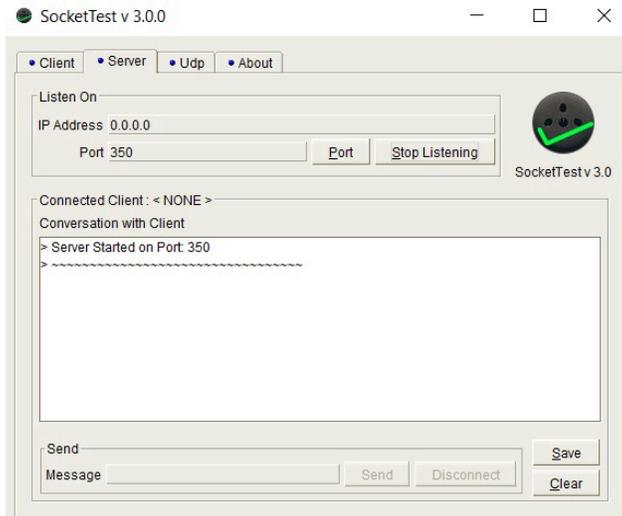
Figure3.4. SocketTest v 3.0.0

4. In order to obtain the IP address of your computer and the default gateway, press Windows R to open "Run" box. Type "cmd" and then click "OK" to open a Command Prompt.

5. In Command Prompt type "ipconfig" to get the IPv4 Address and Default Gateway (i.e., the IP address of the router) in the form of xxx.xxx.xxx.xxx. In our example as it shown below "IPv4 Address = 192.168.1.64" and "Default Gateway = 192.168.1.254"
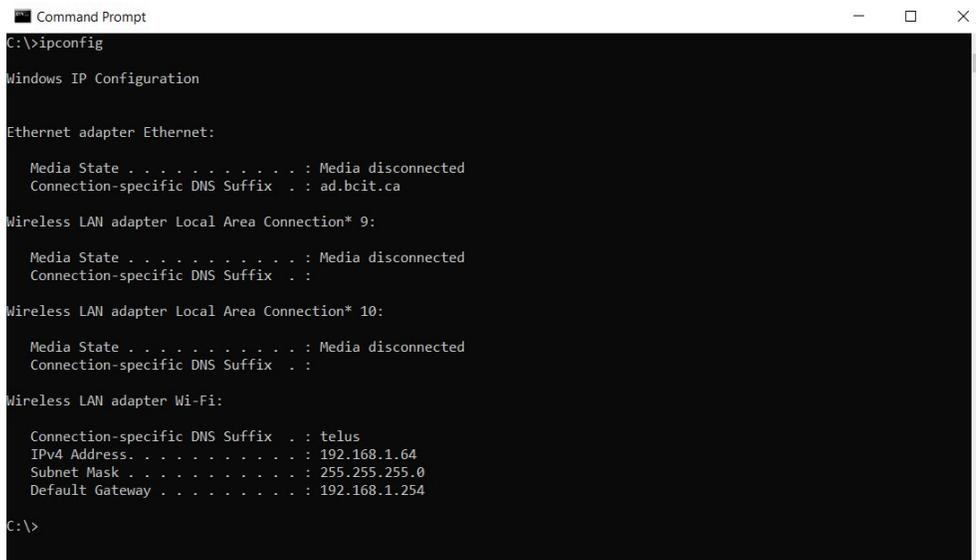
Figure3.5. Command Prompt

6. Go to "Google" and search for "my public IP address". This is the IP address of your ISP router available to the rest of the Internet. Alternatively, you can go to https://canyouseeme.org/ and find your public IP address.
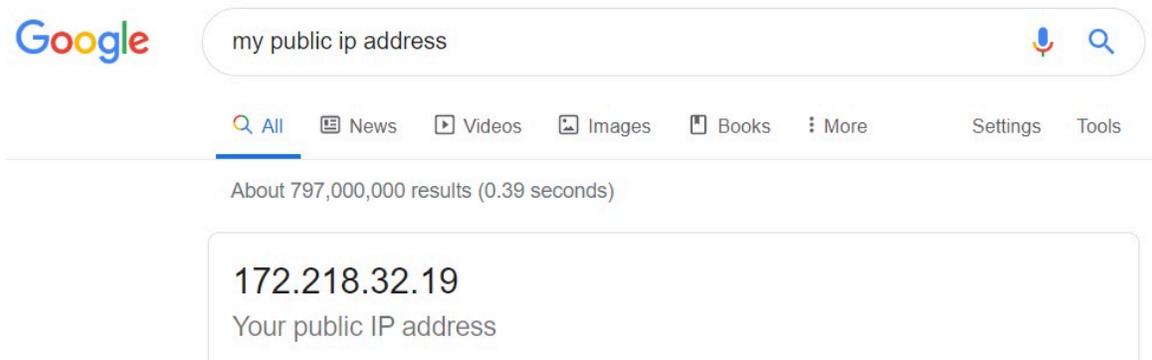
Figure3.6. Public IP address

The Default Gateway is the IP address of the interface of your ISP router connected to your PC. IPv4 address is the private IP address given to your PC by the router. Since the private IP address is not addressable from outside your network, using "port forwarding", you can tell your router where to direct traffic for a specific port on your PC. In other words, port forwarding allows a remote computer on a different network (e.g., BG96 module) to connect to your PC on a private network and run a service behind the router.
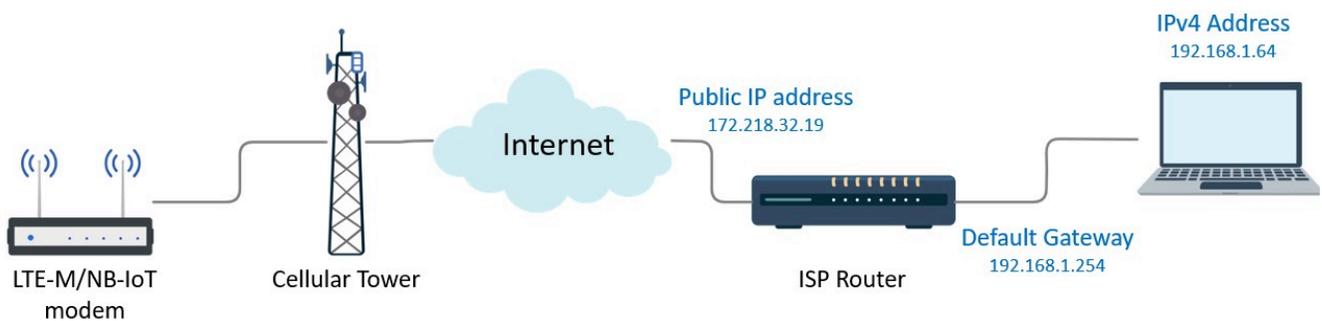
Figure3.7. Network architecture

7. Go to your internet browser and in the address bar type in the "Default Gateway" which was 192.168.1.254 in our example. You may also find the Default Gateway address on your WiFi-Router security tag. Enter the Username/Password for your router to log in. Note that depending on your router you may see a different interface.
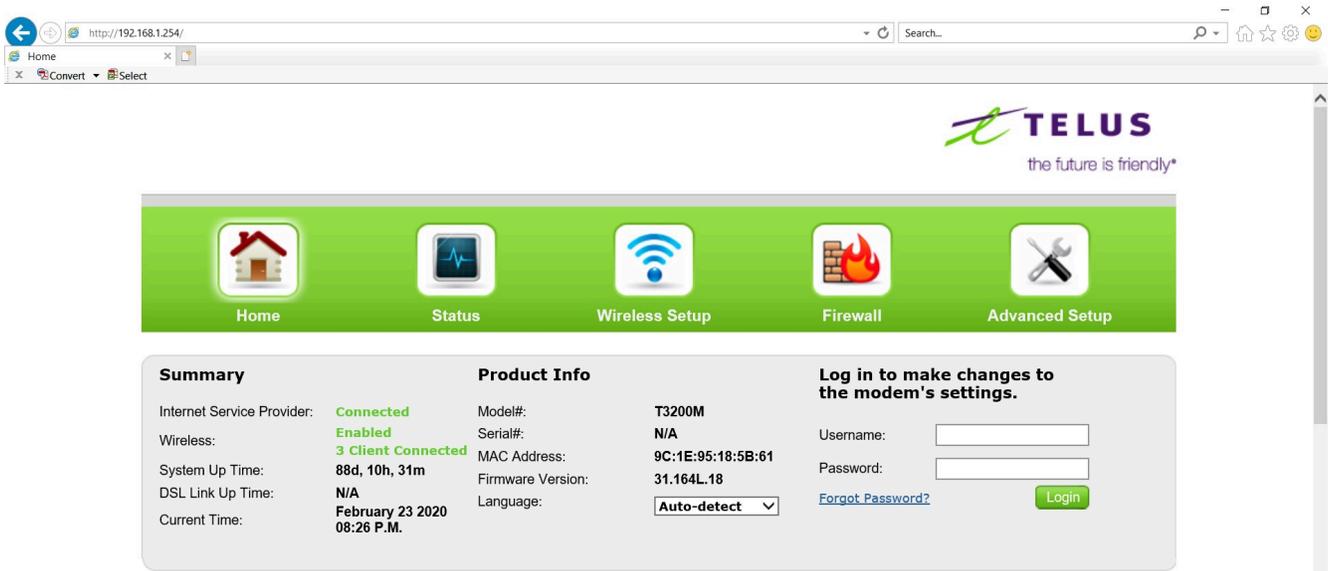
Figure3.8. Router login page

8. To make the service port available to the Internet, you need to set up port forwarding using the IPv4 address (192.168.1.64) and Port number (350) in your router so that your ISP does not block the port as shown in Figure3.9.
9. To verify that the port is open and your ISP does not block it, go to www.canyouseeme.org, enter your public IP address and port number and click "Check Port". Make sure the server you created in SocketTest is listening at that port; otherwise, you will see that the port is closed as shown in Figure3.10.

Figure3.9. Port forwarding example

Figure3.10. Verify port is open

B. Configure BG96 module as TCP/IP client

In this part of the lab, you will set up a TCP client connection to enter into buffer access mode. BG96 TCP/IP AT Commands Manual[2] has the full list and description of different modes and commands required for TCP/IP connections.

10. Open "Tera Term" and use the APN provided by the SIM carrier (e.g., m2m-west.telus.iot) to configure a context via AT Commands.

AT QICSGP=1,1,"M2M-WEST.TELUS.IOT","","",1   *// Set the your mobile carrier APN*

OK

---

11. Activate the context and query the state of the context and IP address of the module. **AT QIDEACT=1** can be used to deactivate a context.

AT QIACT=1   *// Activate a context*

OK

AT QIACT?   *// query the context state*

QIACT: 1,1,1,"10.200.109.166"

OK

---

12. Set up a TCP client connection using your PC public IP address (e.g., 172.218.32.19) and port number (e.g., 350). If the connection is successful " QIOPEN: 0,0" will be returned and SocketTest will show the new client IP address.

---

2. https://www.quectel.com/UploadImage/Downlad/
Quectel_BG96_TCP(IP)_AT_Commands_Manual_V1.0.pdf

```
AT QIOPEN=1,0,"TCP","172.218.32.19",350,0,0    // Set up a TCP client connection using your
own public IP address

OK


QIOPEN: 0,0
```

C. Send and receive data using BG96 module

13. Enter the AT Command below and type in your text. Then press CTR Z to send it. The received text should be shown on SocketTest.

```
AT QISEND=0                // Send changeable length data. After text press CTR Z to send.

> Hello, this is a test

SEND OK
```

14. When you reply using SocketTest, the module will show QIURC: "recv",0 which means a message has been received in the buffer. To read the buffer, type in **AT QIRD=0,1500**.

```
QIURC: "recv",0              // Send changeable length data. After text press CTR Z to
send.

AT QIRD=0,1500

QIRD: 30

Hi, I received your message.


OK
```

15. Close the connection using **AT QICLOSE=0**.

The instructions above explained how to establish a TCP client connection using the BG96 module. If you are interested, referring to BG96 TCP/IP AT Commands

Manual[3] you can try establishing UDP client, TCP server and UDP server connections.

D. Connectivity to the Internet

To check the connectivity between BG96 board and the Internet, we send ping commands. Since ping uses TCP/IP connection, the success of the command below shows that your board is capable of making TCP/IP connection and you can check the connectivity.

AT QPING=1, "www.bcit.ca"

3. https://www.quectel.com/UploadImage/Downlad/
Quectel_BG96_TCP(IP)_AT_Commands_Manual_V1.0.pdf

Chapter 4: Data Communication Protocols for IoT

4.1 Introduction

Massive IoT interconnects an enormous number of devices and services over the Internet for a broad range of applications. The IoT devices are typically connected to the Internet via Internet Protocol (IP).

The application-layer protocols such as HyperText Transfer Protocol (HTTP) are primarily used for communication on browser-based clients, which typically run on relatively high-capability devices, such as smartphones that use relatively high-bandwidth communications channels (e.g., LTE). On the other hand, IoT protocols are set of rules suitable for IoT applications to ensure that data sent from an IoT device gets read and understood by another device. IoT protocols also need to ensure optimum security of the exchanged information between connected devices. Depending on the IoT devices and their applications, there are different application-layer IoT protocols. The application layer is actually an interface between the user and the IoT device. A specific IoT use case may require a specific application-layer data communication protocol and an IoT-centric communications stack.

There are many IoT protocols available to choose for different applications and requirements. For example, some of these protocols are specifically designed to satisfy the requirements for fast and reliable business transactions or some other are optimized to meet the requirements of data collection such as sensor updates in constrained networks. Some other IoT protocols are suitable for applications where Instant Messaging (IM) and online presence detection is needed [5]. Each of these protocols have their own advantages and disadvantages when dealing with

different IoT scenarios. This is why it is important to understand the characteristics of various IoT protocols to be able to choose the best-fil protocol for your use case.

Three widely accepted and key IoT protocols include Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), and Advanced Message Queuing Protocol (AMQP). We first introduce each of these three protocols in detail and then present a general comparison between them to gain insight into the strengths and limitations of each protocol.

4.2 Message Queue Telemetry Transport (MQTT) Protocol

MQTT is a lightweight and scalable publish/subscribe IoT messaging protocol that can be easily implemented to relay data through a central server called the broker. This protocol is primarily used for constrained networks that provide connections to remote locations and is ideal for small devices that require efficient bandwidth and battery use such as car sensors, smoke detectors, smart watches, and text-based messaging applications. Compared to HTTP, MQTT is faster, has less overhead and power consumption. The other difference to HTTP is that in MQTT, a client does not have to pull the information it needs, but if there is new data to be sent, the server (broker) pushes the information to the client.

In MQTT, new devices can be easily added without touching the existing infrastructure (multiple publishers and multiple subscribers). Since the new devices only communicate with the broker, they do not need to be compatible with the other devices.

Figure4.1. MQTT protocol for cellular IoT

The broker acts like a post office. Instead of a device, sending the messages to or getting them from another device directly (peer-to-peer), the client (publisher) sends the messages to the post office (broker) and then it is forwarded to everyone who needs the message (subscribers). The difference is that MQTT uses subject line called "topic" instead of addresses i.e., everyone who needs a copy of that message subscribes to that topic. A topic is a simple string that can have more hierarchy levels, which are separated by a slash (e.g. sensors/temperature). Note that topics are case sensitive.

In MQTT, the IoT devices are interconnected via the broker, meaning that there is no direct connection between clients and the broker relays the messages between clients. Unlike most communication protocols, this message relay is not restricted to a one-to-one model. The message uses a topic for unique identification and is not targeted into a single device. Hence, it is possible to have many devices subscribe to the same topic, which can realize a one-to-many model. Similarly, a many-to-many model multiple devices can be publishing the same topic and have one or multiple subscribers. The broker handles all of these scenarios by using topics to manage the messages.

MQTT is a bi-directional communication protocol where each client can both produce and consume data by publishing messages and subscribing to topics. The big advantage of this two-way communication is that the IoT devices can send sensor data and at the same time receive configuration information and control commands.

A. MQTT Connection

MQTT uses three levels of Quality of Service (QoS) to ensure its message reliability. Table4.1 shows these three QoS levels including fire and forget (Level 0), acknowledged delivery (Level 1), and assured delivery (Level 2).

Table4.1. MQTT QoS levels

QoS	Model	Reliability	Description
0	Fire and forget	Unreliable	Message is delivered without duplication; no acknowledgment is required
1	At least once	Reliable	Message is delivered at least once with possible duplications; acknowledgment is required
2	Exactly once	Reliable without duplications	The message is delivered without duplications

MQTT requires reliable, lossless, and in-order delivery of packets, and thus, it is a TCP/IP-based protocol where a TCP connection needs to be established prior to transferring the data. MQTT is a three-phase handshake communication that needs to first establish the TCP connection. Second, the MQTT connection is established and data is published. Finally, the TCP connection is terminated.
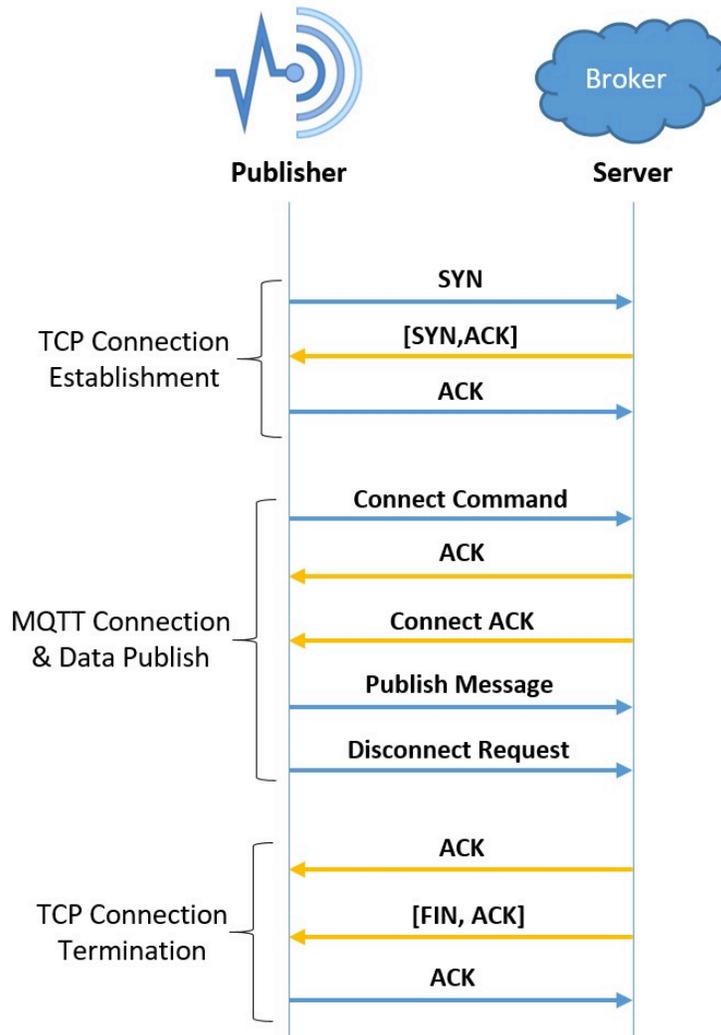
Figure4.2. MQTT publisher-broker handshake

TCP connection establishment

A three-way handshake is used to establish the TCP connection. First, the client sends **SYN** request to indicate that it wants to communicate with the server (broker) which is always in the listening mode. Then, this request is acknowledged by the server by sending a **[SYN, ACK]** packet to let the client know that the server has received the connection request packet and is ready to exchange information with the client. Finally, the client sends **ACK** to make the server aware that the client has received the acknowledgment of connection request and is ready to transfer data with the server.

MQTT connection establishment and data publish

There are different MQTT publisher handshake processes for different QoS levels. For QoS level 0, after the TCP connection is successfully established, the publisher sends the connect request packet to the server (broker). Once the connection request packet is received by the server, it sends an acknowledgment packet to the publisher to show that the broker is ready for communication. Then, the publisher starts publishing the message. Finally, the publisher disconnects from the server by sending a disconnect request packet.

TCP connection termination

After the server receives the disconnect request packet, it sends two packets to the client. One **ACK** packet to acknowledge the reception of disconnect request packet, and the second **[FIN, ACK]** packet to let the client know that the broker also wants to terminate the connection. The last packet in this phase is an acknowledgement sent by the client.

B. Security of MQTT protocol

Most IoT devices lack the necessary network resources to manage their connections in a secure manner. The central broker in the MQTT architecture increases the security by decoupling the devices and applications. In this architecture, all devices have to authenticate against a central security location (the broker) and only one port is required to be open to the broker (the secure port 8883). MQTT provides this security using Transport Layer Security (TLS) encryption, and username/password protected connections. Moreover, each client is unaware of IP addresses and domains of other devices. These security mechanisms are configured on the MQTT broker and then the client will need to comply with the security requirements. This is why the capabilities of the MQTT client should be taken into account when planning and implementing security measures on the broker.

There are three ways that a broker can verify the identity of an MQTT client:

- **Client IDs:** All MQTT clients must provide a Client ID, which is used to link the MQTT topic to the client and to the TCP connection when a client subscribes to a topic.
- **Usernames and passwords:** Before a connection is established, the MQTT broker can require a valid username and password from the client. Note that this username/password combination is not encrypted or secured and is simply transmitted in plain text.
- **Client Certificates:** This most secure method to authenticate the clients is typically very difficult to implement because the certificates need to be deployed and managed on many clients. This authentication method could be suitable for small number of clients with strictly high security needs.

C. Transport Layer Security (TLS)

TLS or as it is more commonly known Secure Socket Layer (SSL) technology is a part of TCP/IP protocol (not MQTT) and provides an encrypted pipe to protect all parts of MQTT messages including the payload. This technology needs to be supported by the client as well and may not be available on cheap and simple IoT devices. Note that data can be encrypted end-to-end at the application layer without configuring the broker. However, passwords cannot be protected because the broker is not involved.

D. MQTT broker

Hosting an MQTT broker

There are two main options to host an MQTT broker. The MQTT broker can be hosted on a locally installed or a cloud-based server. In the former, MQTT broker is installed on one's server hardware. Depending on the requirements of each broker,

one can choose from many free and open source MQTT brokers. Table4.2 below shows some of these available brokers.

Table4.2. Locally installed brokers

Broker	Description
Mosquito	Most popular and lightweight open source broker written in C. Default broker for edge networks.
Mosca	Based on Node.js (an open source server environment) and not very feature rich when compared to Mosquitto. Very simple broker and ideal for small home network deployments. It is installed as a **Node-RED node** and then added to a flow. Node-RED is a flow-based development environment developed by IBM for visual programming and putting together different online services, hardware devices, and APIs as part of the IoT [6].
emqttd	It is an open source and highly scalable broker written in Erlang programming language.

Many companies such as Google, Amazon, Microsoft, and IBM provide cloud-based MQTT server/brokers. For example, Google Cloud IoT Core provides the MQTT protocol through running a managed broker linking to the port "mqtt.googleapis.com:8883". Note that Port 8883 is the standard TCP port reserved secure MQTT connections.

Reliability of MQTT broker

In MQTT, the broker is the central point of communication between all devices, which raises concern about having a single source of failure. To increase the reliability of the communication, most broker software and clients support automatic handover to a backup/redundant broker if the primary server fails. The software can also be set up to share the load of clients across multiple servers on cite, in the cloud, or a combination of both.

Client status certificates

In MQTT, the network load is reduced by having the client send data only when there is a change in the values. The broker retains the published messages and sends them out to new subscribing clients. The MQTT brokers can be aware of the state of their clients and communicate that state to other clients using birth and

death certificates and the Last Will and Testament (LWT) feature. MQTT clients can register an LWT message that will be sent by the broker if they suddenly disconnect. These messages can be used to let the subscribers know when a device disconnects. When a publishing client connects to a broker, it issues a birth certificate along with its LWT payload to the broker. The broker will notify any subscribing client to the same topic as the publishing client whenever the publishing client is offline or online. To ensure that the stale data is not delivered to the subscribing client if the publishing client is offline, the broker provides the subscribing client with the LWT payload. This is accomplished using a pre-configured 'keep alive timer' or 'heart beat check' between broker and publishing clients. As a result, the subscribing clients are always aware whether the publishing client is online, or when the client went offline, and what was the last known value.

4.3 Constrained Application Protocol (CoAP)

The Constrained Application Protocol (CoAP) is an application-layer protocol developed by the Internet Engineering Task Force (IETF) as an extremely lightweight communications protocol stack suitable for resource-constrained devices. CoAP can be implemented over User Datagram Protocol (UDP) and is designed for devices with limited capacity to connect in Light-Weight Machine-to-Machine (LWM2M) communication. LWM2M enables remote management and control of IoT devices using a streamlined managed objects model and provides interfaces for securely monitoring and administering devices. CoAP uses the GET, PUT, POST, and DELETE methods and response codes similar to, but not exactly like, HTTP. Therefore, it is easy to map CoAP traffic from devices to the Representational State Transfer-ful (RESTful) Application Program Interface (API) logic. A RESTful API is an API that utilizes HTTP requests to GET, PUT, POST and DELETE data. An API for a website is a code that allows two software programs to exchange information between them.

CoAP uses the resource model mapped to the Universal Resource Identifier (URI) instead of MQTT topics. However, there is a similarity between the CoAP URIs and

MQTT topics. For example, sensor devices publishing their sensor information to a server could be described in the following manner:

CoAP sensor publishing to a CoAP server

URI: coap://devices/sensors/temperature

MQTT client publishing to a sensor queue on the broker

topic: "/devices/sensors/temperature"

In simple terms, the operation of CoAP can be summarized as follows. A UDP packet is transmitted to request data from the other end-point (a GET on a device URL). Next, a response packet including the requested information (e.g., the temperature value of a sensor) is sent back. Note that a packet of data can also be pushed to a device – a POST to its URL.

CoAP and HTTP protocols have many similar features with the difference that CoAP is optimized for IoT and more specifically for M2M. CoAP has low overhead and is very simple to parse. It has proxy and caching capabilities and exchanges messages asynchronously.

As it is shown in Figure4.3, CoAP is made of two different layers: messages layer which is the lowest layer and deals with UDP, and request/response layer which manages request/response interactions.
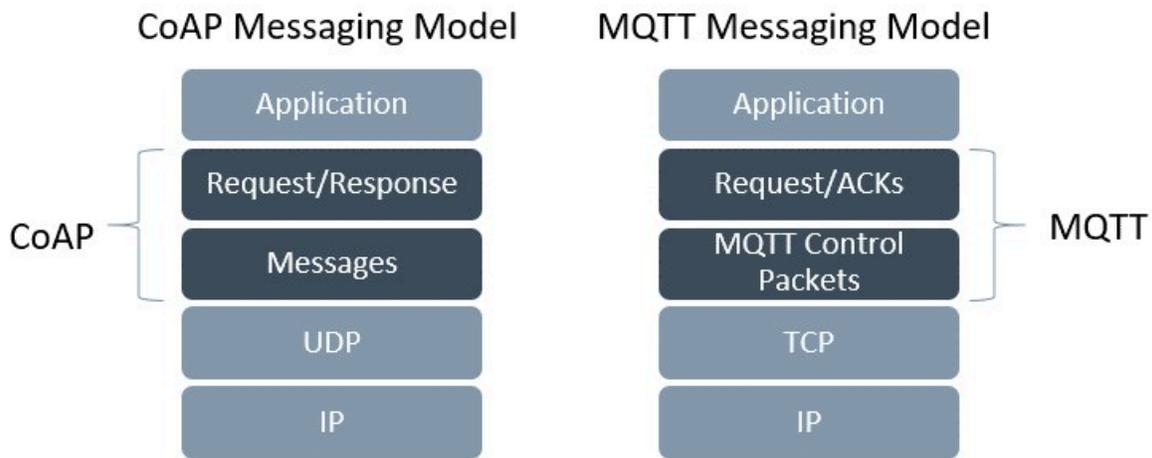
Figure4.3. CoAP and MQTT messaging models

A. CoAP Messages Layer

The messages layer sits on top of the UDP and deals with exchanging messages between the IoT devices and the Internet. The message duplicates are detected using the unique IDs assigned to each message. CoAP provides its own reliability mechanism using confirmable messages and non-confirmable messages.

Confirmable message (CON)

A confirmable message (CON) is a reliable message where the client keeps sending the message using a default timeout and a back-off mechanism between retransmissions until an acknowledge message (ACK) with the same ID is received from the server. The ACK messages can carry useful payload data and themselves do not need separate ACKs.
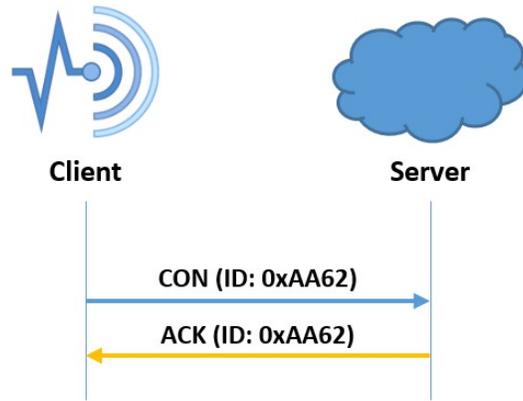
Figure4.4. Confirmable message

In case the server has trouble processing the CON message (e.g., when the server cannot even provide an error response), it replies with an RST message (reset message) instead of the ACK signal.
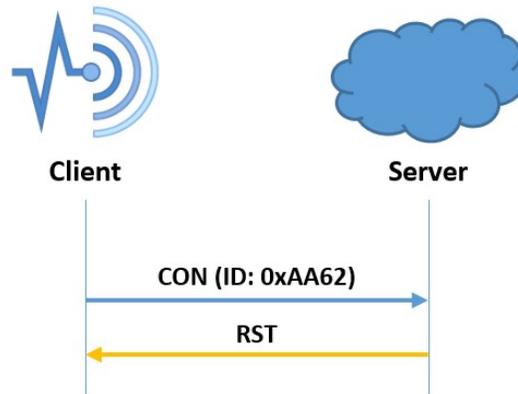
Figure4.5. RST message

Non-confirmable messages (NON)

When exchanging non-critical messages, unreliable NON messages can be used where the server does not acknowledge the arrival of the messages. Although the NON messages are not acknowledged, they are assigned message IDs to detect the message duplicates.
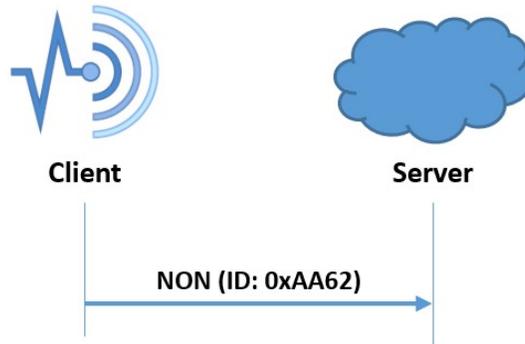
Figure4.6. Non-confirmable message

CoAP unlike MQTT does not provide an explicit QoS level model. However, the confirmable/non-confirmable message types can be mapped to the MQTT QoS semantics. Table4.3 shows CoAP and MQTT QoS mapping.

Table4.3. CoAP and MQTT QoS mapping

Delivery	CoAP	MQTT
Unreliable	CON (Confirmable)	QoS Level 0
Reliable	NON (Non-confirmable)	QoS Level 1

B. CoAP Request/Response Layer

This layer can send the request using either a CON or NON message. In a scenario where the request is sent using a CON message, an ACK message containing the response or the error code is responded by the server provided that it can answer immediately. In such a communication scheme, the request and the response are matched using a token (this token is different than the message ID).
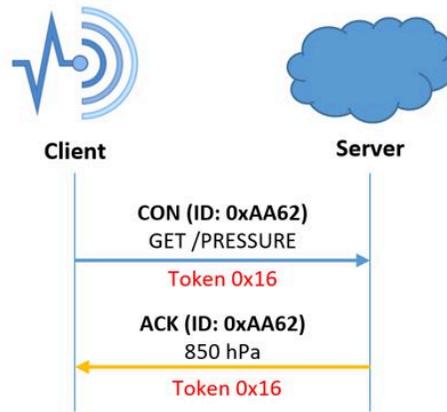
Figure4.7. Request/response using CON messages when the server's response is available immediately

In case the server cannot respond right away, it sends an ACK message with no content as the response (empty response). Once the response is ready, a new CON message containing the response is transmitted to the client. After receiving the response, the client will acknowledge that. Note that the server's response will also be a NON message, if the request sent by the client is a also a NON message.
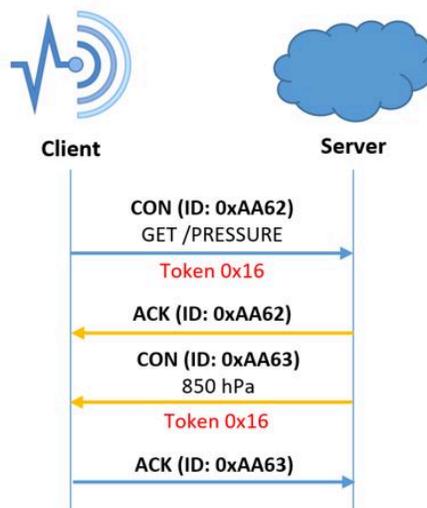
Figure4.8. Request/response using CON messages when the server's response is not available immediately

C. Multicast Group Communications

Group communications (one-to-many or multicast) which allow a single request to be sent to many devices are supported by CoAP.

D. Observers

Observers reduce the need for constant polling by allowing a client to receive updates based on the state of the subject. In other words, the state of a smart object is dependent on the states of the other devices that it is connected to. An example of observers in action is when a motor reacts to a sensor measuring the rotational speed of a generator, by adjusting its speed accordingly. The observer client's status keeps updating until either a CON message is unacknowledged or an explicit removal message is sent by the client, or the last notification is expired.

E. CoAP Security

Same as HTTP, CoAP is also a plaintext protocol by default which means that to secure the communication, an additional encrypted protocol as a wrapper is required. In CoAP the data encryption is done by Datagram Transport Layer Security (DTLS) over UDP to secure and protect the information.

4.4 Advanced Message Queuing Protocol (AMQP)

AMQP is a lightweight application layer protocol optimized for higher security and reliability, easier provisioning and interoperability. AMQP protocol supports both publish/subscribe (such as MQTT) and request/response (such as CoAP) architectures. AMQP is a connection-oriented protocol (the client and the broker need to establish a connection before they transfer data) because it uses TCP as its transport protocol. AMQP is a considered a reliable protocol with two levels of QoS

for message delivery including unsettle format (similar to MQTT QoS 0) and settle format (reliable similar to MQTT QoS 1).

4.5 Comparative analysis of IoT protocols

As it was seen in the previous sub-sections each IoT protocol has its own data transfer model. For example comparing the two IoT protocols we discussed earlier in this chapter, MQTT and CoAP use publish/subscribe and request/response models, respectively. In MQTT, a central broker is used to forward the received messages from the publisher to the subscribers (MQTT can support multiple publishers and subscribers for the same topic). On the other hand, similar to the HTTP, CoAP is basically a one-to-one protocol. MQTT is an event-oriented protocol (the client will publish a topic whenever there is an update) while CoAP is more suitable to transfer the state of the device. The main difference between CoAP and MQTT results from the MQTT requirement to run on the reliable, lossless, in-order, byte-stream delivery transport, which effectively mandates the TCP/IP stack. CoAP, on the other hand, runs on top of UDP and provides its own reliability mechanism using confirmable and non confirmable message types. Overall, MQTT/TCP traffic requires approximately 10x more transactions and the data overhead is approximately 100x higher compared to the CoAP/UDP traffic for the 5-byte payload. The characteristics of the IoT protocols have been summarized in Table 4.4 [7].

Table4.4. Characteristics of the IoT protocols

	MQTT	CoAP	AMQP	HTTP
Base protocol	TCP	UDP	TCP	TCP
Paradigm	Publish/Subscribe	Request/Response or Publish/Subscribe	Publish/Subscribe or Request/Response	Request/Response
Header Size	2 Bytes	4 Bytes	8 Bytes	Undefined
Message Size	Small and Undefined (up to 256 MB)	Small and Undefined	Negotiable and Undefined	Large and Undefined
Reliability	QoS 0 - At most once QoS 1 - At least once QoS 2 - Exactly once	CON Message NON Message	Settle Format Unsettle Format	Limited (via TCP)
Standards	OASIS, Eclipse Foundations	IETF, Eclipse Foundation	OASIS, ISO/IEC	IETF and W3C
Licensing	Open Source	Open Source	Open Source	Free

Depending on the messaging requirements of IoT use cases, different messaging protocols may be selected. This is why it is important to understand the strength and limitations of each protocol to determine their best-fit applications. Table4.5 presents an evaluation and relative analysis of the four widely accepted and used IoT protocols discussed in this chapter. This relative comparison in terms of message size, power consumption, bandwidth, reliability, security, and interoperability and so on, helps to gain insight into the pros and cons of each protocol [7].

When thinking about selecting MQTT and CoAP the following recommendations shall be taken into account:

- For IP data transmission over NB-IoT networks, CoAP protocol is the preferred.
- CoAP can provide from 224 to 1,157 percent higher payload efficiency compared to the TCP/IP-based MQTT. So, it offers superior transmission performance for both network capacity and IoT solutions.
- MQTT over NB-IoT networks is not recommended due to very high overhead and increased number of data transactions.
- MQTT is appropriate for relatively long-standing MQTT/TCP/IP connections that transmit relatively high volumes of data (e.g., telematics applications). For IoT solutions that require MQTT transport, other types of radio technologies, such as LTE, should be considered.

Table4.5. IoT application protocols: comparative analysis

	MQTT	CoAP	AMQP	HTTP
Message size & overhead	Lightweight Small header size: 2 bytes per message. Larger overhead & message size due to TCP requirements.	UDP-based No connection overhead	TCP overheads for connection establishment and tear down.	Most heavyweight protocol due to TCP overheads for connection establishment and tear down.
Power Consumption vs. Resource Requirement	Designed for resource constrained devices.		Requires slightly higher power resources.	Requires more processing power and resources.
Bandwidth vs. Latency	The slow start of TCP to avoid congestion results in less bandwidth utilization.	Reduced network load response times.	Extra services need higher bandwidth and latency.	Requires larger bandwidth and latency time
Reliability & QoS	Guranteed packet delivery.	NON CON and CON messages.	Settle and Unsettle Format	No default QoS.
Interoperability	Only pub/sub	Limited to UDP supported devices.	Serializes the structured data	Most interoperable
Security	Minimal authentication Simple username and password	DTLS and IPsec for encryption and authentication	TLS negotiation: Singleport TLS, Pure TLS and WebSockets Tunnel TLS	HTTP Basic and HTTP Digest.
IoT usage	Facebook, IBM, Cisco, Red Hat, AWS	Erika , Cisco (Field Area Network), Contiki	NASA's Nebula Cloud Computing	Not suitable for IoT industry.
Standardization	Not a global standard. OASIS open standards consortium and Eclipse Foundation	IETF standard Supported by Eclipse Foundation	OASIS adopted international standard ISO/IEC 19464:2014	Global standard for the Web

Chapter 5: MQTT Function

5.1 Introduction

In this chapter, we would like a BG96 board to act as a MQTT client and publish its data with a specific topic to a broker. To build a broker, we use an architecture similar to the one used in Chapter 3. The broker is built on a computer by using Node-RED[1] programming tool to wire together hardware devices, and online services to establish a communication with the BG96 module through MQTT protocol.

The data published on the broker will be sent to all the MQTT clients that have subscribed to the topic. To have another MQTT client, we use a MQTT client software on the same computer. The configuration used in this lab is shown in Figure 5.1. The MQTT client on the computer also can publish a message to MQTT broker that can be forwarded to BG96 board.
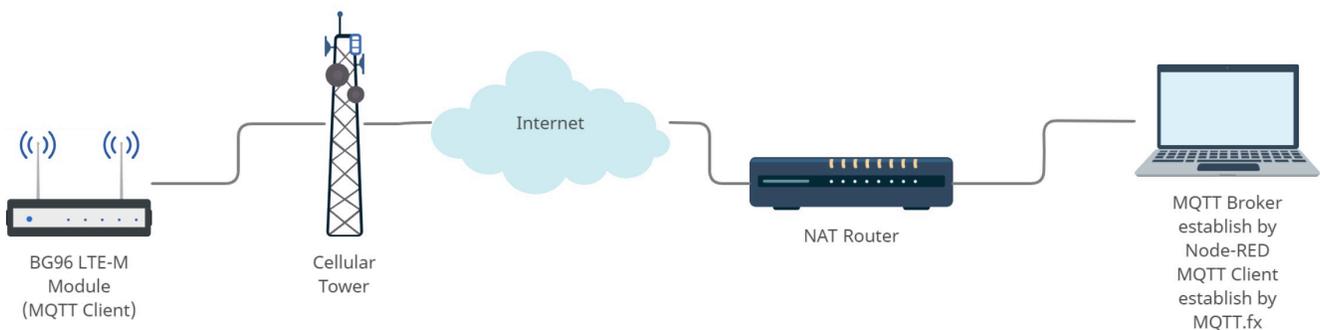
Figure5.1. MQTT broker and clients using the BG96 module

1. https://nodered.org/

5.2 MQTT Broker

In Project A of this lab, we try to build the MQTT-broker and a MQTT client on a computer. In project B, we set up the BG96 board to become a MQTT client and publish the data. Since the computer is behind a NAT router and uses internal IP address, the router needs to have been setup properly and we should do port forwarding for Port 1880 and 1883.

A. Install and run Node-RED

1. Download and install the recommended version of the JavaScript runtime Node.js[2]
2. Open Command Prompt and type in the following code to install Node-RED as a global module including its all dependencies:

    npm install -g –unsafe-perm node-red

3. Once installed, in the Command Prompt type in **Node-red** to start Node-RED. If you want to stop Node-RED you can press Ctrl-C or close the terminal window.
4. Access the Node-RED editor by entering http://localhost:1880 at your browser's address bar. To get familiar with Node-Red, you can watch Intro to Node-RED: Part 1 fundamentals[3].
5. Go to Settings -> Palette->Install and search for "node-red-contrib-mqtt-broker" to install Mosca MQTT broker (server).

2. https://nodejs.org/en/#home-downloadhead
3. https://www.youtube.com/watch?v=3AR432bguOY
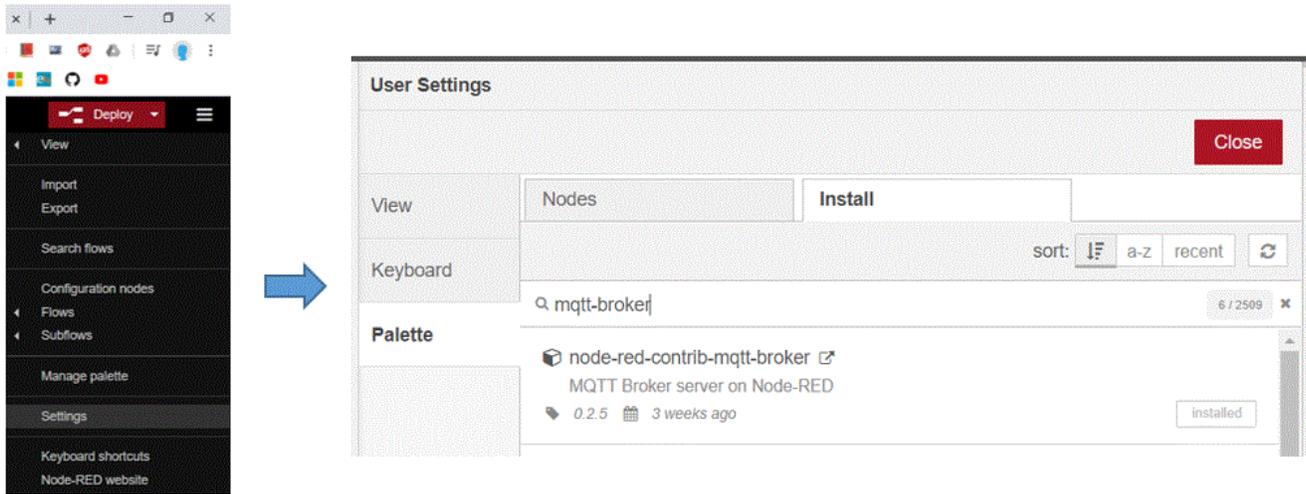
Figure5.2. Installing Mosca MQTT broker

6. From the left side pallet drag and drop the following nodes to form the MQTT communication and set the settings according to Table5.1.

Table5.1. Node description and settings:1

Node	Description	Required settings
inject	Injects a message into the flow either manually or at regular intervals.	Payload: timestamp
debug	Displays selected message properties in the debug sidebar.	Output: msg.payload
mqtt in	Connects to a MQTT broker and subscribs to messages from the specific topics.	Name: public broker Server: localhost Port: 1883 Topic: sensor
mqtt out	Connects to a MQTT broker and publishes messages	Name: public broker Server: localhost Port: 1883 Topic: sensor
mosca in	MQTT broker	MQTT port: 1883

7. Connect the nodes as shown below. Before deploying the flow, using the same port forwarding process as you did in the previous lab, make sure that you have access to Port 1880 and 1883. To make sure that everything is set up correctly, you should download and install MQTT.fx[4] to use as a MQTT client.

4. https://mqttfx.jensd.de/
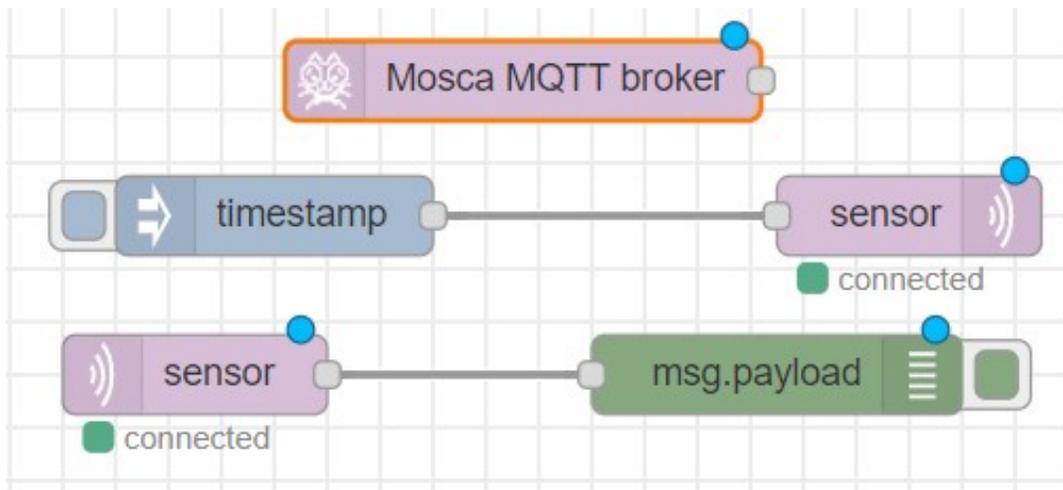
Figure5.3. MQTT clients and broker nodes interconnection

B. MQTT function of Quectel BG96 module through AT commands

In Lab 2, you learned how to obtain the public IP address of your computer (e.g., 172.218.32.19). Also, as it was shown in Table5.1, the MQTT port was set as 1883 (TCP/IP port 1883 is reserved with Internet Assigned Numbers Authority (IANA) for MQTT) and the topic was chosen as "sensor", i.e., any device that subscribes to this topic will receive the messages published with this topic. If you refer to BG96 MQTT Application Note[5], you can find the AT Commands to publish/subscribe to a specific topic.

8. Replace your own public IP address and enter the following AT Commands. You should be able to see the message in the debug slide bar of Node-RED (press the button to access).

5. https://www.quectel.com/UploadImage/Downlad/
   Quectel_BG96_MQTT_Application_Note_V1.1.pdf

```
AT QMTOPEN=0," 172.218.32.19 ",1883    // Open a network for MQTT client. Use your own IP

OK


QMTOPEN: 0,0                            //Opened the MQTT client network successfully

AT QMTCONN=0,"MyClientID"               // Connect a client to MQTT server. Use any client
ID

OK



QMTCONN: 0,0,0

AT QMTPUB=0,0,0,0,"sensor"             // Publish a message to topic "sensor"

>Hello MQTT. This is my message to publish.    // Press CTR Z to send

OK



QMTPUB: 0,0,0
```

9. Enter the following AT Command to subscribe the BG96 module to the "sensor" topic. Then, in Node-RED, inject timestamp or any string. You should see the received message on the BG96 module. You may use **AT QMTDISC=0** AT Command to disconnect the client from the MQTT server.

```
AT QMTSUB=0,1,"sensor",2

OK


QMTSUB: 0,1,0,1

QMTRECV: 0,2,"sensor","1583885887680"    // This shows the received timestamp


AT QMTDISC=0


QMTDISC: 0,0
```

10. From the left side palette and under the "storage" nodes, drag and drop the "file" node. This is where you can store the data received from module. You can add a "function" node to format the message and add a timestamp before storing the message in your file. Also, install the "Email" node, then drag and drop it. You will use this node to send a notification if a specific message is received (e.g., if the temperature is above a certain number or if the string is "Hello"). Table5.2 summarizes the description and required settings for these three nodes. If you are using Gmail, go to your "account preferences" make sure that "Allow less secure apps" is enabled.

Table5.2. Node description and settings:1

Node	Description	Required settings
function	A JavaScript function block to run against received messages.	msg.time = new Date().toLocaleString(); msg.payload={"time":msg.time,"Temp":msg.payload}; return msg;
file	Writes msg.payload to a file.	Enter the destination and file name e.g., C:\MyProject\MyFile.txt
switch	Routes messages based on their property values or sequence position	e.g., Msg.payload==Hello
email	Sends the msg.payload as an email with a subject of msg.topic	To: YourEmail@address.com Userid: SourceEmail@address.com

The flow looks like the one depicted in Figure5.4.
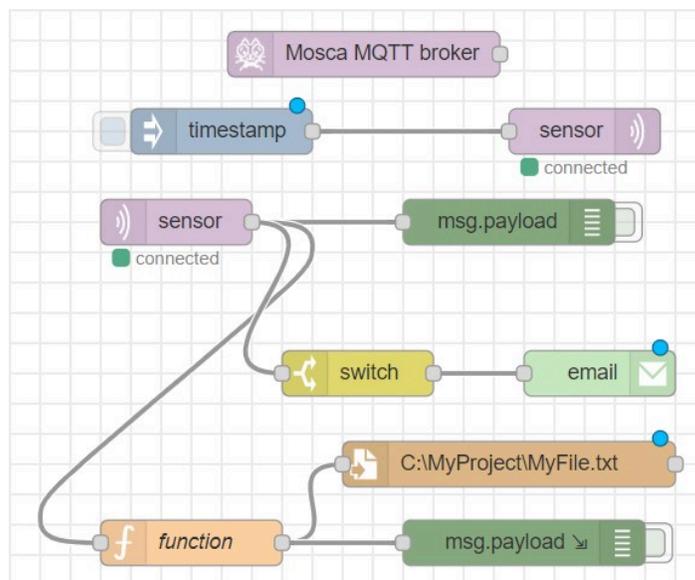
Figure5.4. The flow with all necessary nodes

11. Go to Hamburger Menu -> Export to generate the "*.json" file for future use.

Chapter 6: Microsoft Azure IoT Hub

6.1 Introduction

Microsoft Azure IoT Hub is a cloud-based managed service acting as a central message hub for two-way secure communication between the IoT application and the IoT devices that it manages. IoT Hub supports several messaging patterns such as file upload from devices, device-to-cloud telemetry, and request-reply methods to control the IoT devices from the cloud. IoT Hub monitoring helps to maintain the health of the IoT solution by tracking events such as device failures, creation, and connections [8].

6.2 Microsoft Azure IoT Hub

In this chapter, you will become familiar with Microsoft Azure IoT Hub and create an account. Then, you will register the IoT device with Microsoft Azure IoT Hub to exchange messages.

A. Microsoft Azure account

1. Go to azure.microsoft.com and create a free Microsoft Azure account.
2. Go to the Azure portal[1] and follow the instructions[2] on Microsoft website to create an IoT hub using the Azure portal.

---

1. https://portal.azure.com/#home
2. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-create-through-portal
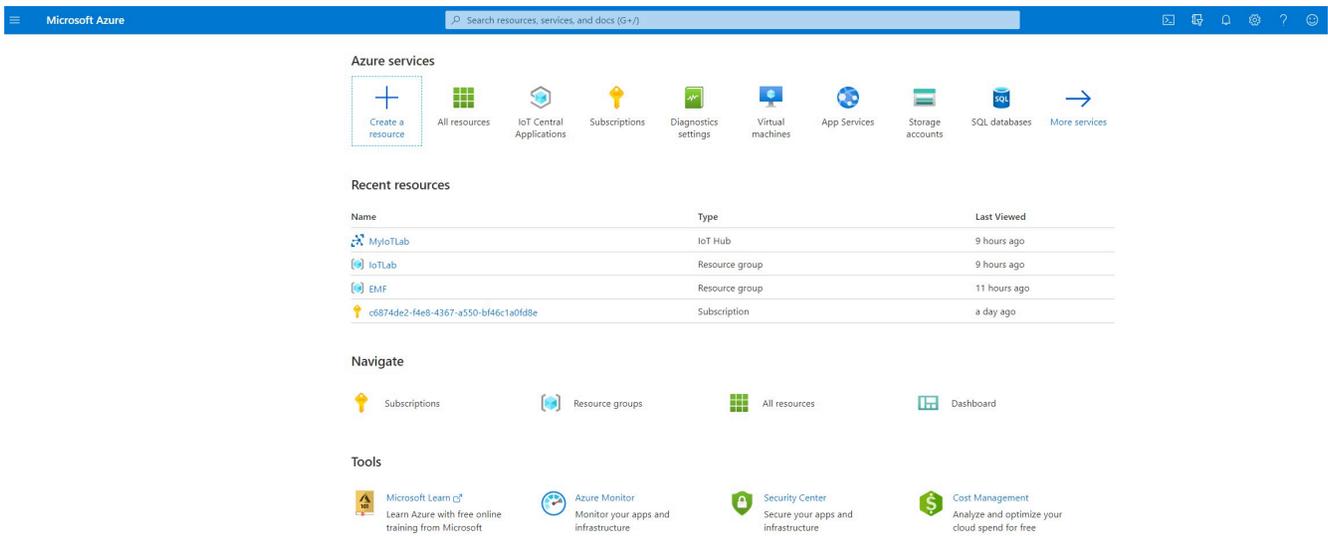
Figure6.1. Microsoft Azure portal

In this example and through the procedure to create an IoT hub, we created a new resource group called "IoTLab" and entered "MyIoTLab" as the IoT hub name. In the "size and scale" tab, make sure you select the free pricing tier to avoid extra charges to your bill.
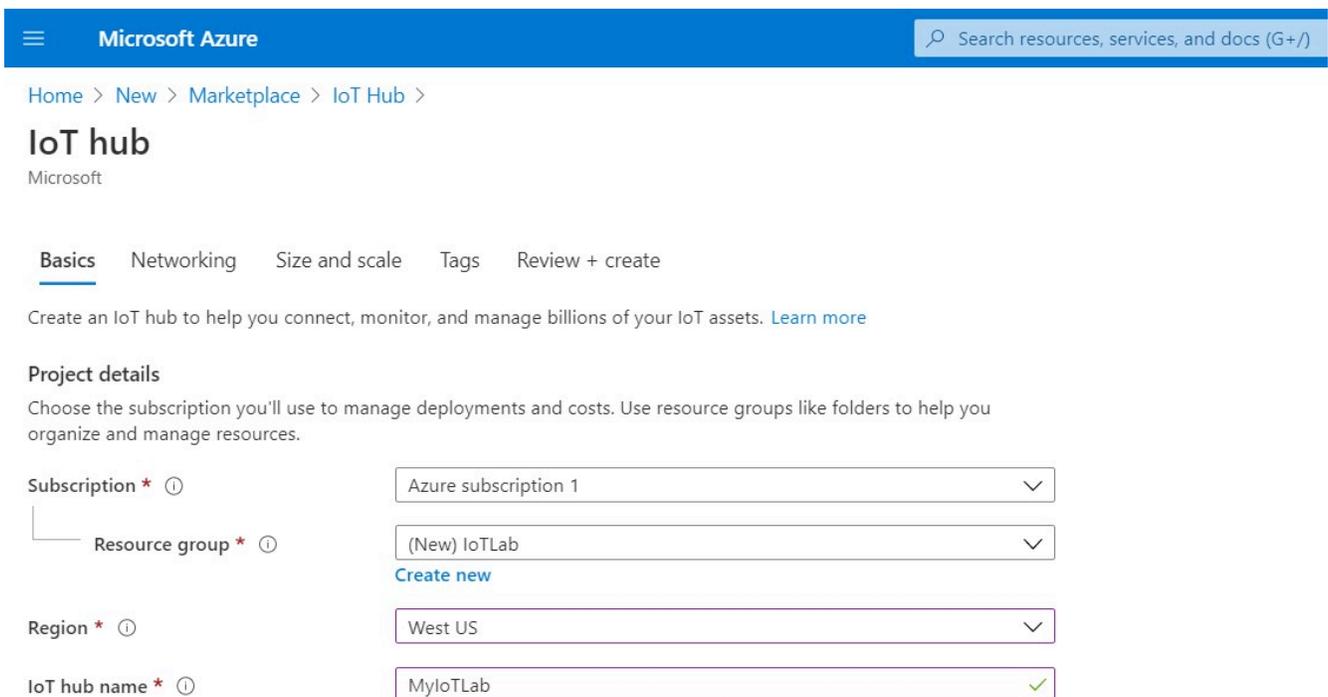
Figure6.2. Create IoT hub – basic tab

3. Use the instructions[3] to register a new device in the IoT hub. Once the device is created, open the device from the list in the **IoT devices**pane and take note of the **Primary Connection String** (it will be used later). This string gives you three important pieces of information for your newly registered device including HostName, DeviceID, and SharedAccessKey that will be used later when configuring the nodes in Node-RED. For example, as you see in the picture below, the primary connection string for the new device that we created is:

HostName=MyIoTLab.azure-devices.net;
DeviceId=MyDeviceID;
SharedAccessKey=W/uBNVjPNq1ntkPpv8t5xCyMyVOdy7YjrO IAVCtFi8=

Therefore, in this case:

- HostName = MyIoTLab.azure-devices.net
- DeviceID = MyDeviceID
- Device SharedAccessKey = W/uBNVjPNq1ntkPpv8t5xCyMyVOdy7YjrO IAVCtFi8=
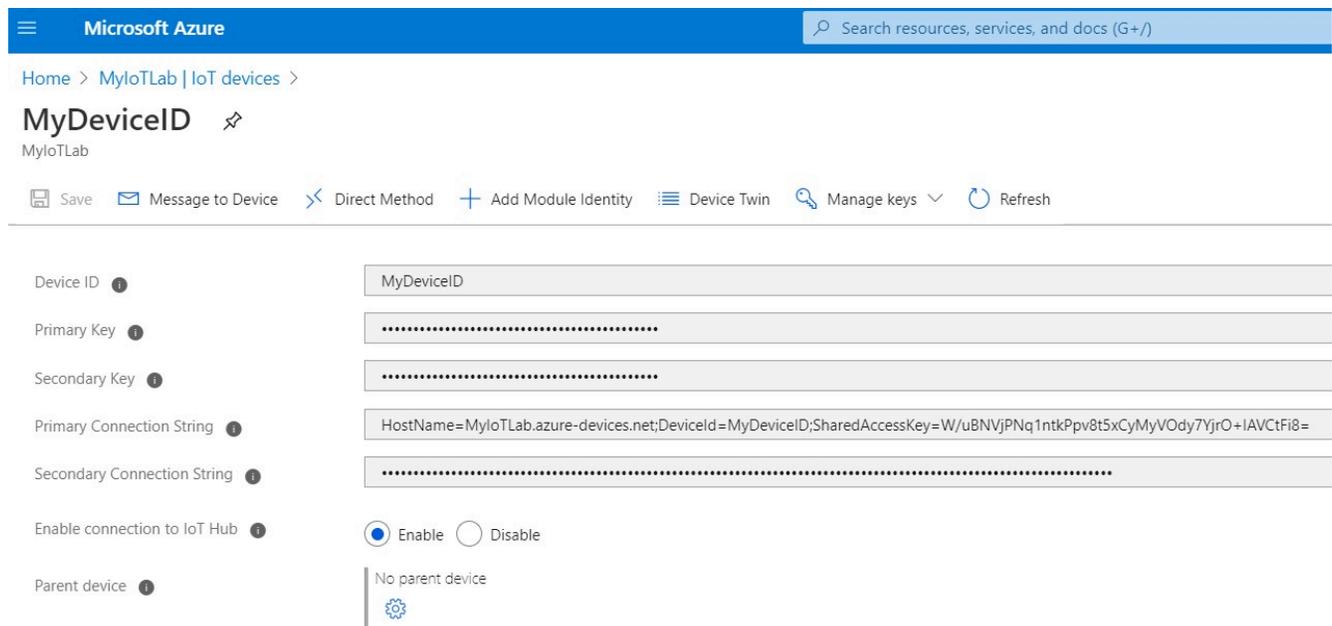
Figure6.3. Device registry information

3. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-create-through-portal

4. In the previous step, you registered and obtained the information for your registered device. In order to get the access information about your IoT hub, in the IoT portal go to "All resources" and click on your IoT hub that you created. Then, go to Shared access policies -> iothubowner and copy the **Connection string – primary key**. For example, our IoT hub connection string is:

```
HostName=MyIoTLab.azure-devices.net;
SharedAccessKeyName=iothubowner;
SharedAccessKey=8JUL6zD9joqYCQyeBIP9uyhQh6u 8BhjP/mg3u ucRg=
```

As you could see, the HostName is the same as for the device in the previous step; but, we have:

- SharedAccessKeyName = iothubowner
- IoT Hub SharedAccessKey = 8JUL6zD9joqYCQyeBIP9uyhQh6u 8BhjP/mg3u ucRg=
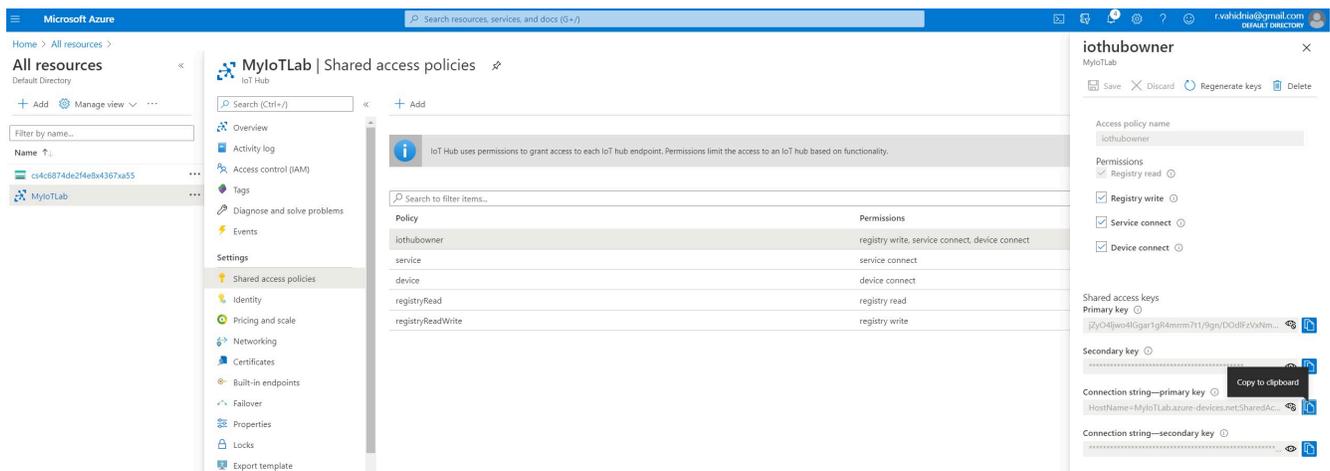
Figure6.4. hared access policies to get the connection string – primary key

B. Device Explorer

The Device Explorer tool is a Windows graphical tool to manage the devices in IoT Hub.

5. Click on Azure IoT SDKs releases[4] and scroll down to the Downloads section to

locate the download link for the **SetupDeviceExplorer.msi**installer. Now download and run the installer.

6. Open the Device Explorer tool. On the connection tab, paste the Shared Access Policies connection string (with HostName, SharedAccessKeyName, and SharedAccessKey) and click Update to connect the Device Explorer to your IoT hub.
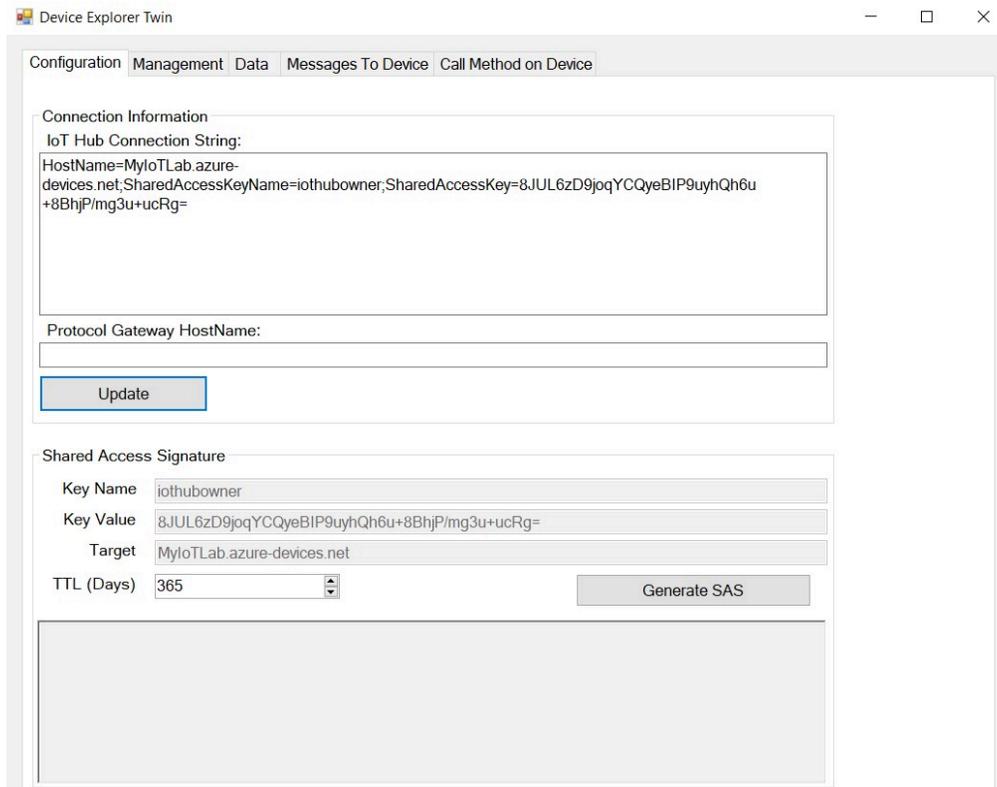
Figure6.5. Device Explorer

7. Click on the Management tab. You can see the device you already created on Azure portal. You can also right click on the device to retrieve the connection string.
8. Go to Data tab and select your event hub and device ID and click on Monitor to start receiving data on Azure IoT hub. Here is where you will see the data received from your IoT device.

4. https://github.com/Azure/azure-iot-sdks/releases
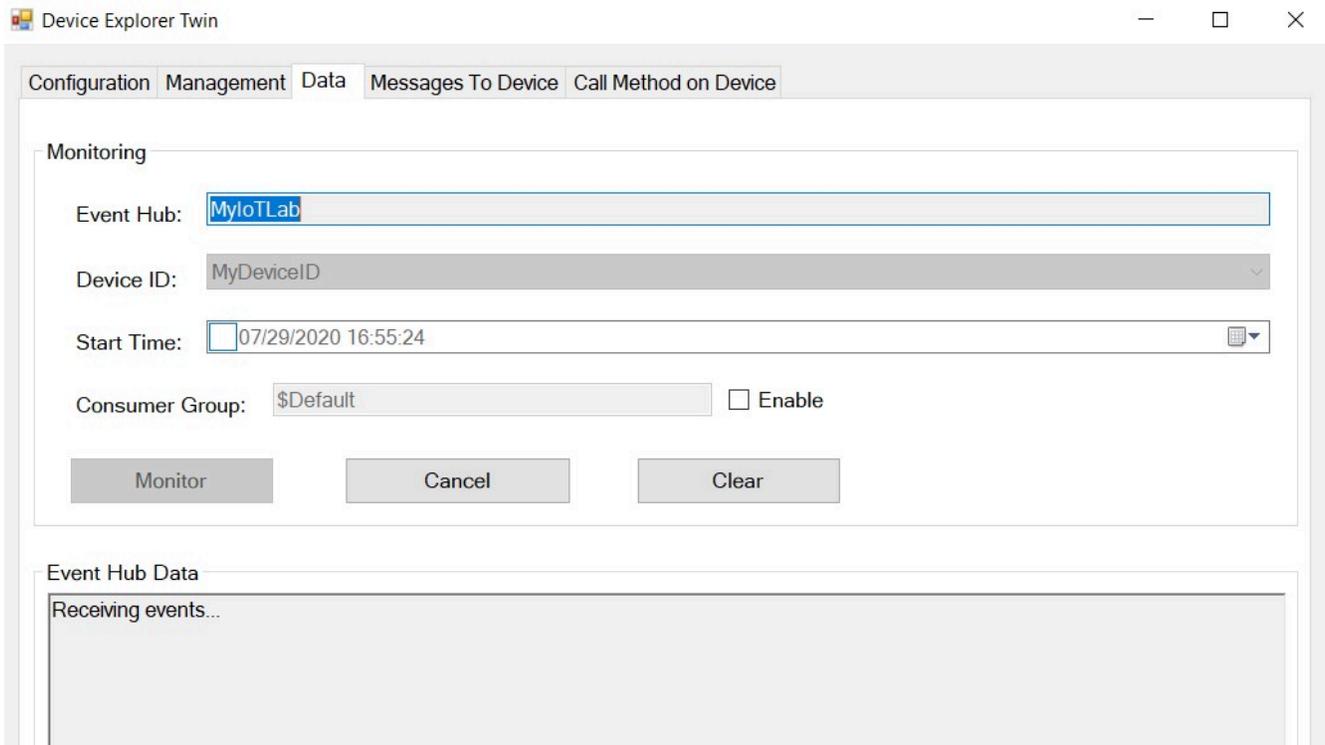
Figure6.6. Device explorer is receiving data from the IoT device

C. Node-RED flow

9. Download and install the recommended version of the JavaScript runtime Node.js[5]. Skip this step if you have Node.js installed on your computer.

10. Open Command Prompt and type in the following code to install Node-RED as a global module with all its dependencies. Skip this step if you have Node-RED installed on your computer.

npm install -g –unsafe-perm node-red

11. In the Command Prompt type in the following command to start Node-RED. If you want to stop Node-RED, you can do so by pressing Ctrl-C or closing the terminal window. *Node-red*

5. https://nodejs.org/en/#home-downloadhead

12. You can then access the Node-RED editor by pointing your browser at http://localhost:1880. Make sure Port 1880 is open.

13. Go to Settings -> Palette -> Install and search for "node-red-azure-iot-hub-mod" to install Microsoft Azure IoT hub nodes.
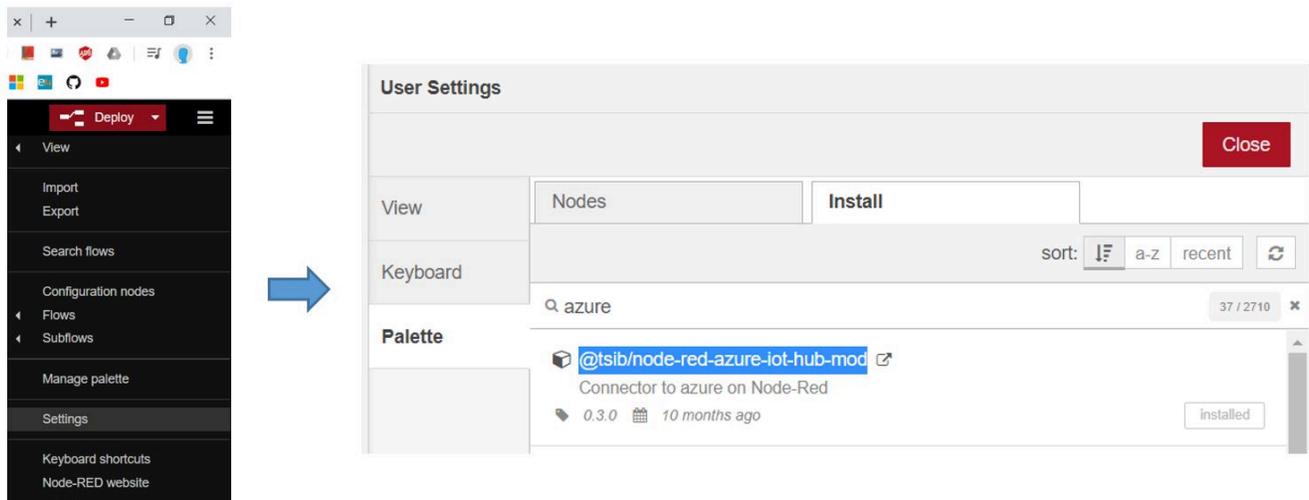
Figure6.7. Installing Microsoft Azure IoT hub nodes

14. Same as previous lab, your IoT device will publish to an MQTT server. You will need a "function" node to prepare the message (i.e., add the device ID and primary key to the message payload) for the Azure IoT Hub.

15. Drag and drop "mqqt in", "mosca in", "debug", "function" and "Azure IoT Hub" nodes into the canvas and connect them as shown below. You will need the information in the **Primary Connection String** to configure these nodes in the next step.
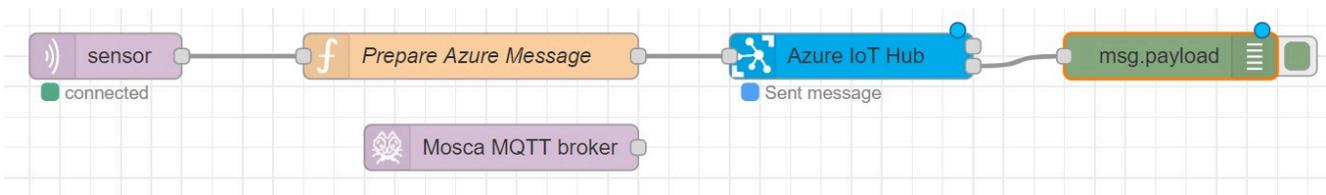
Figure6.8. Nodes interconnection

16. Similar to the previous lab, configure the mqqt nodes so that your IoT device

can publish to the broker. Use the following JSON object and the "DeviceID" and the "SharedAccessKey" to set up the "function" node to prepare the message for the "Azure IoT Hub". Use the "HostName" in the **Primary Connection String**to set up the "Azure IoT Hub" node and pick "mqtt" as the protocol. Before deploying the flow, make sure that you have access to Port 1880, 1883 and 8883 (Azure MQTT port)

msg.payload={

'deviceId':"xxxxxxxxx",

'key':"xxxxxxxxxxx",

'protocol':"mqtt",

'data':msg.payload

}

return msg;

For more information on JavaScript Object Notation (JSON), visit: https://www.w3schools.com/js/js_json_intro.asp
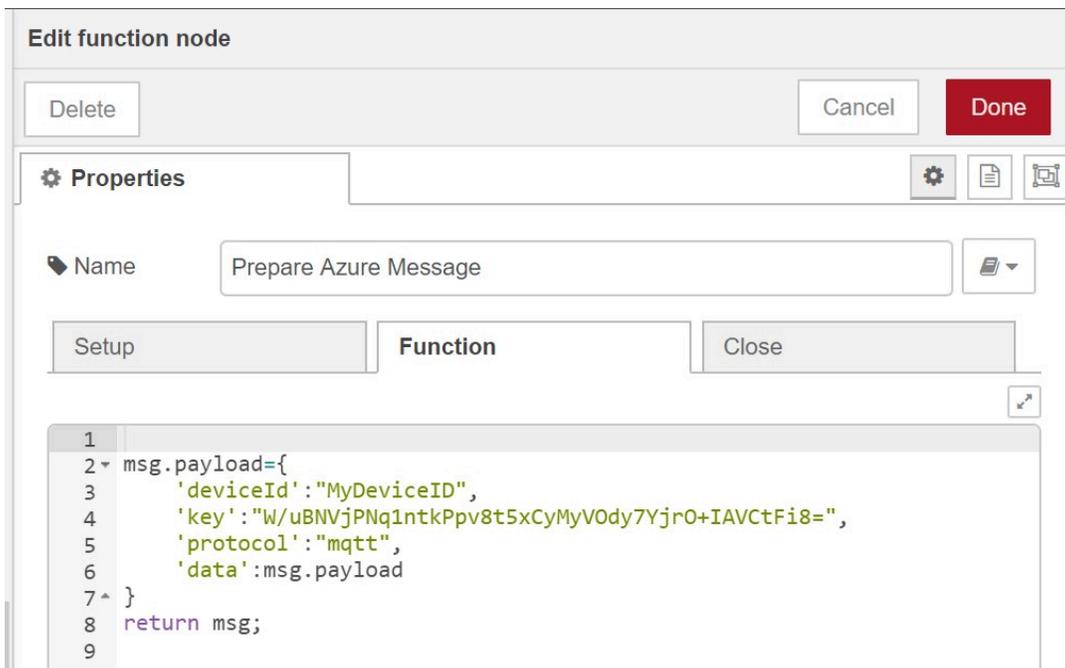
Figure6.9. Prepare Azure message
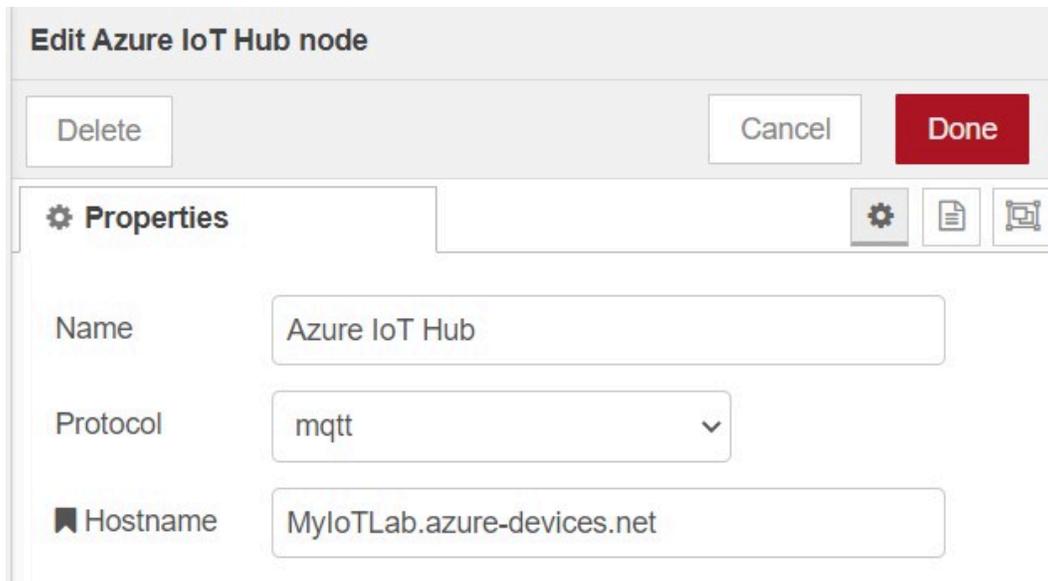
Figure6.10. Azure IoT Hub node settings

Table6.1. Nodes settings
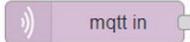
Node	Description	Required settings
mqtt in	Connects to a MQTT broker and subscribs to messages from the specific topics.	Name: public broker Server: localhost Port: 1883 Topic: sensor
mosca in	MQTT broker	MQTT port: 1883
function	A JavaScript function to run against the messages received by the node.	deviceID Key Protocol
Azure IoT Hub	A simple node that sends the message payloads to Azure IoT Hub	HostName Protocol
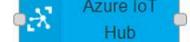
D. Device to Cloud MQTT function

This part of the lab is similar to Lab 3.

17. Replace your own public IP address and enter the following AT Commands. You should be able to see the message "Temp=25" in the data tab of Device Explorer.

```
AT QMTOPEN=0," 172.218.32.19 ",1883    // Open a network for MQTT client. Use your own IP

OK


QMTOPEN: 0,0                            //Opened the MQTT client network successfully

AT QMTCONN=0,"MyClientID"               // Connect a client to MQTT server. Use any client
ID

OK


QMTCONN: 0,0,0

AT QMTPUB=0,0,0,0,"sensor"              // Publish a message to topic "sensor"

>Temp=25                                // Press CTR Z to send

OK


QMTPUB: 0,0,0
```

Figure6.11 shows how the received data will be represented in Device Explorer tool.
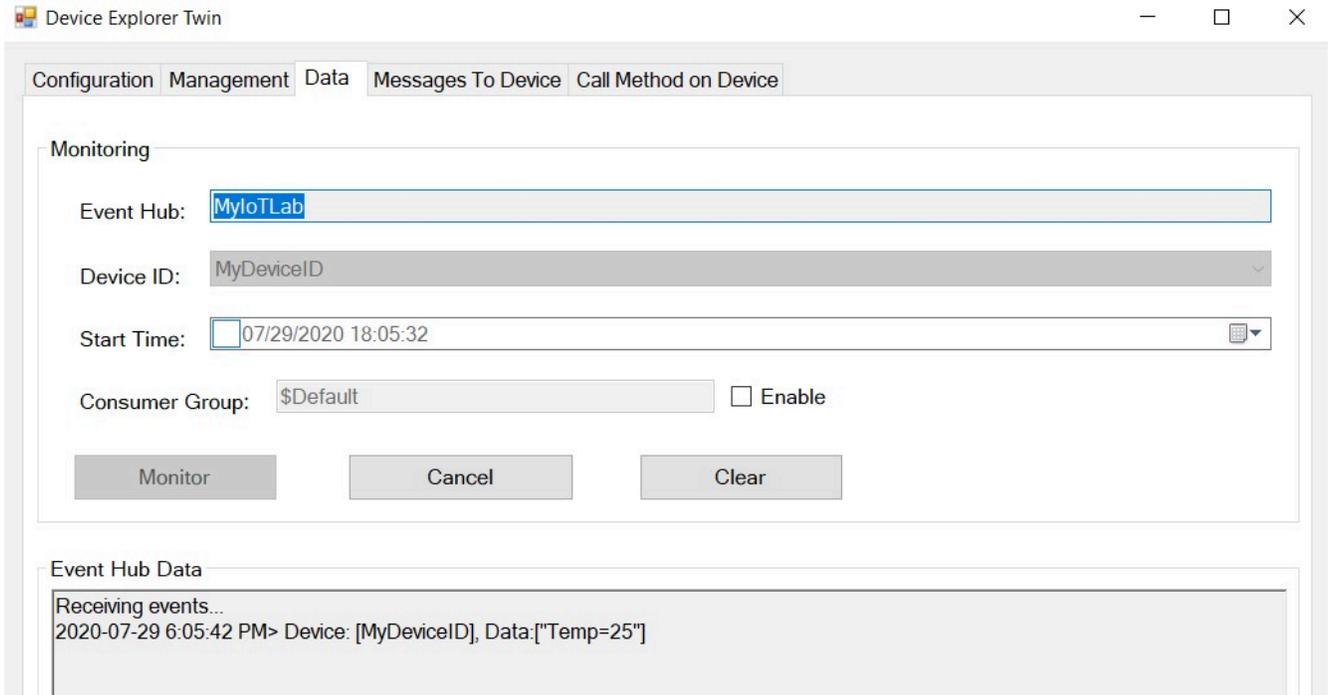
Figure6.11. Received message on the IoT hub

E. Cloud to Device MQTT function

18. Make appropriate changes to your Node-RED flow and enter the AT command (same as Lab 3) so that the IoT device can subscribe to the MQTT broker with topic "sensor".

19. In the Device Explorer tool go to "Message to Device" tab. Type and send a message to your device. You should see this message in your IoT device as well as the debug pannel in the Node-RED environment. You may need to use the "function" node to format the received message.
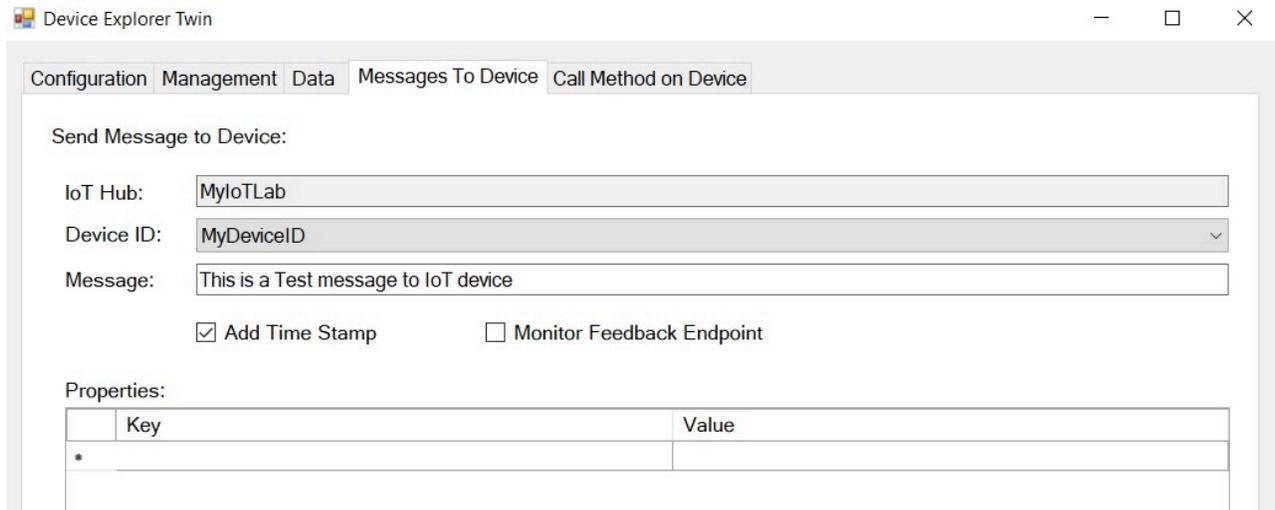
Figure6.12. Cloud to Device messaging

By conducting this lab you were able to communicate with Microsoft Azure IoT Hub. Now, if you are interested, you can make yourself familiar with other features of Microsoft Azure to create a dashboard and visualize the data, store the data on the cloud or send email alerts.

Abbreviations

A

APN   Access Point Name

ACK   Acknowledgement

AMQP   Advanced Message Queuing Protocol

AWS   Amazon Web Services

API   Application Program Interface

B

BER   Bit Error Rate

C

CIoT   Cellular IoT

CON   Confirmable

CoAP   Constrained Application Protocol

D

DTLS   Datagram Transport Layer Security

DL   Downlink

E

EEG   Electroencephalogram

ERP   Enterprise Resource Planning

eDRX   Extended Discontinuous Reception

G

GNSS   Global Navigation Satellite System

H

HTPP   HyperText Transfer Protocol

I

IM   Instant Messaging

ICCID   Integrated Circuit Card Identifier

IMEI   International Mobile Equipment Identity

IANA   Internet Assigned Numbers Authority

IETF   Internet Engineering Task Force

IoT   Internet of Things

IP   Internet Protocol

J

JSON   JavaScript Object Notation

L

LWT   Last Will and Testament

LWM2M   Light-Weight Machine-to-Machine

LPWA   Low Power Wide Area

M

MQTT   Message Queue Telemetry Transport

N

NB-IoT Narrow Band Internet of Things

NAT   Network Address Translation

NON   Non-confirmable

P

PDP   Packet Data Protocol

PSM   Power Saving Mode

Q

QoS   Quality of Service

R

RAT   Radio Access Technologies

RSSI   received signal strength indicator

S

SSL   Secure Socket Layer

T

TCP   Transport Control Protocol

TLS   Transport Layer Security

U

URI   Universal Resource Identifier

UL   Uplink

UDP   User Datagram Protocol

UE   User Equipment

References

[1]   IoT Analytics, "IoT Platforms: The central backbone for the Internet of Things," 2015.

[2]   R. Barton, G. Salgueiro, J. Henry, D. Hanes, P. Grossetete, "IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things," 1st Edition, Cisco Press, 2017.

[3]   F. John Dian, and Reza Vahidnia, "IoT Use Cases & Technologies," ISBN978-1-990132-00-1, 2020.

[4]   F. J. Dian and R. Vahidnia, "LTE IoT Technology Enhancements and Case Studies," IEEE Consumer Electronics Magazine, vol. 9, no. 6, pp. 49-56, Nov 2020.

[5]   N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," IEEE International Systems Engineering Symposium (ISSE), pp. 1-7, 2017.

[6]   [Online]. Available: nodered.org.

[7]   N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in IEEE International Systems Engineering Symposium (ISSE), Vienna, 2017 .

[8]   [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-hub/about-iot-hub.

[9]   "Cellular and LPWA IoT Device Ecosystems, IoT Research Series," 2018.

[10]   "Leading the LTE IoT evolution to connect the massive Internet of Things," Qualcomm, 2018.