Let Us Build a MQTT Pub-Sub Client In C# For IoT Research

Sudip Chakraborty [1] & P. S. Aithal [2]

[1] D.Sc. Researcher, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India,
OrcidID: 0000-0002-1088-663X; E-mail: drsudip.robotics@gmail.com

[2] Senior Professor, Srinivas University, Mangalore, India,
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

Area/Section: Computer Science.
Type of the Paper: Experimental Research.
Type of Review: Peer Reviewed as per |C|O|P|E| guidance.
Indexed in: OpenAIRE.
DOI: https://doi.org/10.5281/zenodo.10603409
Google Scholar Citation: IJMTS

How to Cite this Paper:
Chakraborty, S. & Aithal, P. S. (2024). Let Us Build a MQTT Pub-Sub Client In C# For IoT Research. *International Journal of Management, Technology, and Social Sciences (IJMTS)*, *9*(1), 104-114. DOI: https://doi.org/10.5281/zenodo.10603409

International Journal of Management, Technology, and Social Sciences (IJMTS)
A Refereed International Journal of Srinivas University, India.

CrossRef DOI: https://doi.org/10.47992/IJMTS.2581.6012.0334

Received on: 20/01/2024
Published on: 02/02/2024

Let Us Build a MQTT Pub-Sub Client In C# For IoT Research

Sudip Chakraborty [1] & P. S. Aithal [2]
[1] D.Sc. Researcher, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India,
OrcidID: 0000-0002-1088-663X; E-mail: drsudip.robotics@gmail.com
[2] Senior Professor, Srinivas University, Mangalore, India,
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

ABSTRACT

Purpose: *MQTT stands for Message Queuing Telemetry Transport. It is a lightweight protocol specifically designed for IoT applications. Nowadays, most IoT projects exchange sensor data over the MQTT protocol. It is simple to integrate and can run on low hardware resources. To test MQTT, the researcher needs MQTT publisher subscriber client software. There are several free and paid software available on the web. But sometimes, researchers need some custom interface or functionality not available in the free version. The paid version demands vast amounts of money for customization. Occasionally, they want to avoid customization for specific projects. Here, we provide a procedure to create our MQTT pub-sub client software interface, which the researcher can easily customize. The project is available to download.*
Methodology/Approach: *We installed Microsoft Visual Studio in our working system. Using C# language, we create a GUI (graphical user interface). Inside the GUI, we segregate the info into two. The left is for publishing clients, and the right is for subscribing clients. We installed the M2MQTT package using the NuGet package manager to communicate with the MQTT broker.*
Findings/Result: *Using our built application, we test to exchange the sensor data between two clients. We found the data exchanged in almost real-time. In a couple of scenarios, we observed that the data propagation could have been faster when we set the update interval below 500 milliseconds. There may be a network delay, or the MQTT broker we used is a free service. It may be the limited capacity of the accessible mode of the server we used. If the application is not time-critical, we can utilize this communication route to exchange the sensor data.*
Originality/Value: *Every day, new researchers are introduced to the IoT field and integrate MQTT into their projects. They feel one customizable software they can use according to their project requirements. So, using this project, the researcher can fulfill their need. The code is freely available on the web. And scope to customize. This project provides value to them.*
Type of Paper: *Experimental-based Research.*
Keywords: MQTT pub-sub client in C#, MQTT protocol in C#, MQTT Debugger in C#

1. INTRODUCTION :

Problem statement: The IoT is well known now. The popular protocol in the field of IoT is MQTT. To debug the sensor data transmission, we need an MQTT debugger. Plenty of free and paid software versions are available to debug the protocol over the net. Our researcher mainly uses the accessible version of the MQTT debugger. There are several limitations available inside that application. According to our experience, sometimes our researchers need extra information from the application interface for their research work. So, we realized that we needed to build our custom mqtt debugger, fully customized to our project's needs. Here, we demonstrate such a custom MQTT GUI for the researcher.
Indication of methodology: We used C# language with a Desktop form application to build the software. A model view controller is the best architecture for building a GUI. The paper [14] provides a detailed description of how to build complete software using MVVM architecture. The MVVM is the

best for large-scale projects. We make this utility software in C# desktop applications to better understand the new researcher. On the GUI, we added a couple of user controls. We add the M2MQTT package using the Nuget package manager to handle the MQTT communication.
Essential findings of others in this field: Much research is carried out in IoT, especially the MQTT protocol. Nowadays, this protocol is the backbone of the IoT. The paper (Velinov et al. (2019). [1]) demonstrates the MQTT-based Internet of Things. There is a lot of ongoing research work. Need benchmarking. Longo et al. (2022) [2] research the benchmarking framework for distributed MQTT brokers. Hästbacka et al. (2022) [3] demonstrate the Interoperability of OPC UA PubSub with existing message broker integration architectures. A couple of sound research studies, like Gültunca et al. (2018) [7], examine and compare the communication protocols on the application layer in IoT. Chakraborty et al. (2022) [11-30] demonstrate the application of IoT in their various research work.
What study is done in this paper: we create an MQTT pub-sub client using c# language. Using the application, we tested the available free MQTT broker. Not all brokers' performance is fast responsive. A few brokers are good and provide near real-time data between pub and sub-client nodes or devices. When we tested the broker performance, we noticed several scopes needing improvement in the available MQTT debugger. We listed most of the features we implemented in this research work. The researcher can try the features and add more to their work using the provided code.
Principal conclusion: several research works are focused on the MQTT protocol, the primary communication of IoT integration. The researcher needs to debug the MQTT protocol-enabled device. They should have a good protocol analyzer to conduct an in-depth study of this protocol. The available software suffers several drawbacks, and it is discouraging to use them. Through this research work, we developed good handy tools for the researchers working on the MQTT protocol.

2. REVIEW OF LITERATURE/ CURRENT STATUS :

An unmeasured amount of research work has been carried out on MQTT. Here, we included some research projects where we found noticeable work already done. Table 1 lists a couple of research works and used technology.

Table 1: The list of research work and used technology

S. No.	Focus/Subject	Technology/Algorithm/ Module/Components	Reference
1	Covert channels in the MQTT-based Internet of Things	IoT, MQTT	Velinov et al. (2019). [1]
2	A benchmarking framework for distributed MQTT brokers	MQTT	Longo et al. (2022). [2]
3	Interoperability of OPC UA PubSub with existing message broker integration architectures	MQTT	Hästbacka et al. (2022). [3]
4	Modeling Distributed MQTT Systems Using Multicommodity Flow Analysis	MQTT	Manzoni et al. (2022). [4]
5	Reflection-based Prototyping Framework for OPC UA Servers for Companion Specifications	OPC UA Servers	Walker et al. (2023). [5]
6	MQTT protocol employing IOT-based home safety system with ABE encryption	IoT, MQTT	Gupta et al. (2021). [6]
7	Examination and comparison of the communication protocols on the application layer in IoT	MQTT, IoT	Gültunca et al. (2018). [7]
8	Investigating messaging protocols for the Internet of Things (IoT)	IoT	Al-Masri et al. (2020). [8]
9	Towards network-assisted publish-subscribe over wide area networks	WAN, MQTT	Chang et al. (2020). [9]

10	A semantic publish-subscribe architecture for the Internet of Things	IoT, MQTT	Roffia et al. (2016). [10]

In the table below, we included several research works based on communication technology. The below research work might be helpful for the new researcher for their research work.

Table 2: lists the author's research on IoT and the MQTT field.

S. No.	Focus/Subject	Technology/Algorithm/ Module/Components	Reference
1	A Practical Approach To GIT Using Bitbucket, GitHub, and SourceTree	Bitbucket and GitHub website. SourceTree windows application	Chakraborty et al. (2022). [11]
2	How to make IoT in C# using Sinric Pro	C# language. Sinric Pro IoT website	Chakraborty et al. (2022). [12]
3	Virtual IoT Device in C# WPF Using Sinric Pro	Sinric Pro IoT website, C# WPF framework	Chakraborty et al. (2022). [13]
4	MVVM Demonstration Using C# WPF	C# MVC framework	Chakraborty et al. (2023). [14]
5	Let Us Create An IoT Inside the AWS Cloud	AWS Cloud	Chakraborty et al. (2023). [15]
6	Let Us Create a Physical IoT Device Using AWS and ESP Module	AWS cloud, ESP Wifi Module	Chakraborty et al. (2023). [16]
7	Let Us Create Multiple IoT Device Controller Using AWS, ESP32 And C#	AWS cloud, ESP32 Wifi Module, C# language	Chakraborty et al. (2023). [17]
8	Let Us Create Our Desktop IoT Soft-Switchboard Using AWS, ESP32 and C#	AWS cloud, ESP32 Wifi Module, C# language	Chakraborty et al. (2023). [18]
9	Let Us Create an Alexa Skill for Our IoT Device Inside the AWS Cloud	AWS Cloud, Alexa developer console	Chakraborty et al. (2023). [19]
10	Let Us Create A Lambda Function for Our IoT Device In The AWS Cloud Using C#	AWS Lambda console, AWS cloud, C# language	Chakraborty et al. (2023). [20]
11	Modbus Data Provider for Automation Researcher Using C#	C# Language	Chakraborty et al. (2023). [21]
12	IoT-Based Industrial Debug Message Display Using AWS, ESP8266, And C#	AWS cloud, ESP8266 module, And C#.	Chakraborty et al. (2023). [22]
13	IoT-Based Switch Board for Kids Using ESP Module And AWS	AWS cloud, ESP8266 module And C# language	Chakraborty et al. (2023). [23]
14	Let Us Create an Alexa-Enabled IoT Device Using C#, AWS Lambda and ESP Module	AWS Lambda console, ESP module, And C# language	Chakraborty et al. (2023). [24]
15	Alexa Enabled IoT Device Simulation Using C# And AWS Lambda	AWS Lambda console and C# language	Chakraborty et al. (2023). [25]
16	CRUD Operation on WordPress Database Using C# SQL Client	WordPress website, C# SQL client module	Chakraborty et al. (2023). [26]
17	CRUD Operation On WordPress Database Using C# And REST API	WordPress website, C#, and REST API	Chakraborty et al. (2023). [27]

18	CRUD Operation on WordPress Posts From C# over REST API	WordPress website, C#, and REST API	Chakraborty et al. (2023). [28]
19	CRUD Operation On WordPress Custom Post Type (CPT) From C# Over REST API	WordPress website, C#, and REST API	Chakraborty et al. (2023). [29]
20	Let Us Build a WordPress Custom Post Type (CPT)	WordPress website and REST API	Chakraborty et al. (2023). [30]

3. OBJECTIVES OF THE PAPER :

(1) To study the implementation procedure of the MQTT protocol pub-sub client using C# language.
(2) Review the performance of the free MQTT broker on the web.
(3) To analyze the response of available MQTT brokers.
(4) To compare the performance of the various free brokers.
(5) To evaluate the performance of our designed application.
(6) To test the protocol efficiency.
(7) To prove that our designed application performs better.
(8) To design valuable tools for the field of MQTT research projects.
(9) To develop excellent handy tools in the IoT communication field.
(10) To interpret the process flow of the MQTT protocol.
(11) To create complete one-stop debug tools for MQTT research.

4. METHODOLOGY :

Figure 1 depicts the project block diagram. The typical scenario is that the MQTT broker is situated in the cloud to be accessible from anywhere. In the C#, to communicate with the MQTT broker, there is a popular library called M2MQTT. If the subscriber is available, it forwards it to the subscriber; if not, it just discards the topic data. On the other side, MQTT subscribers listen to the topic. When data is available, it triggers the call-back function. We read the data from the callback function. The process flow of the subscriber is the same as that of the publisher.

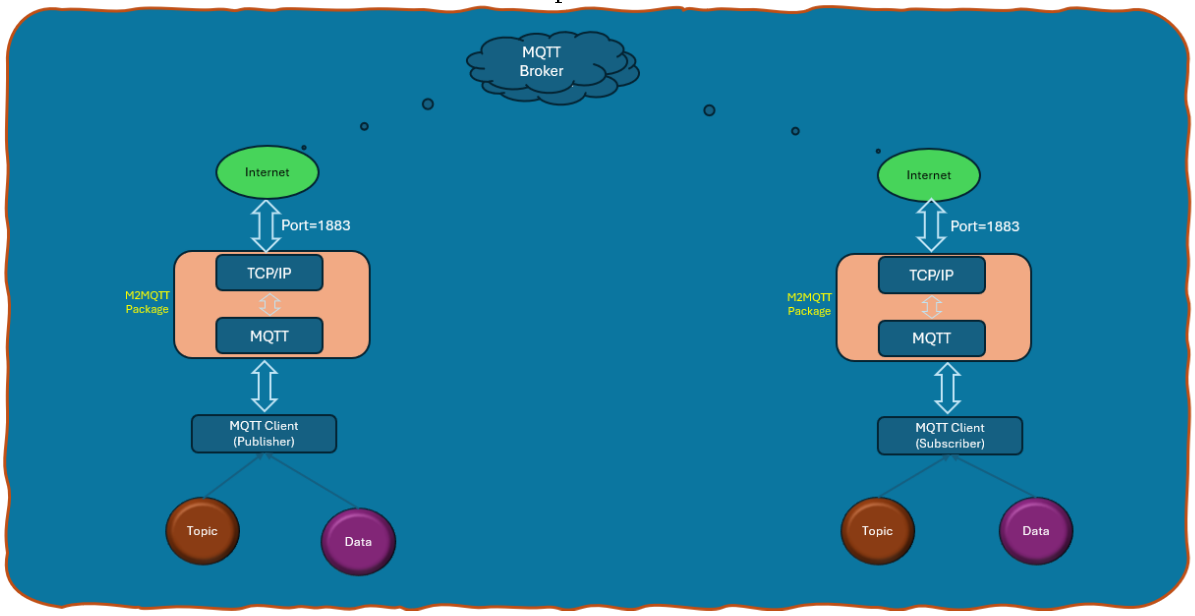


Fig. 1: The Project block diagram (Source: Author)

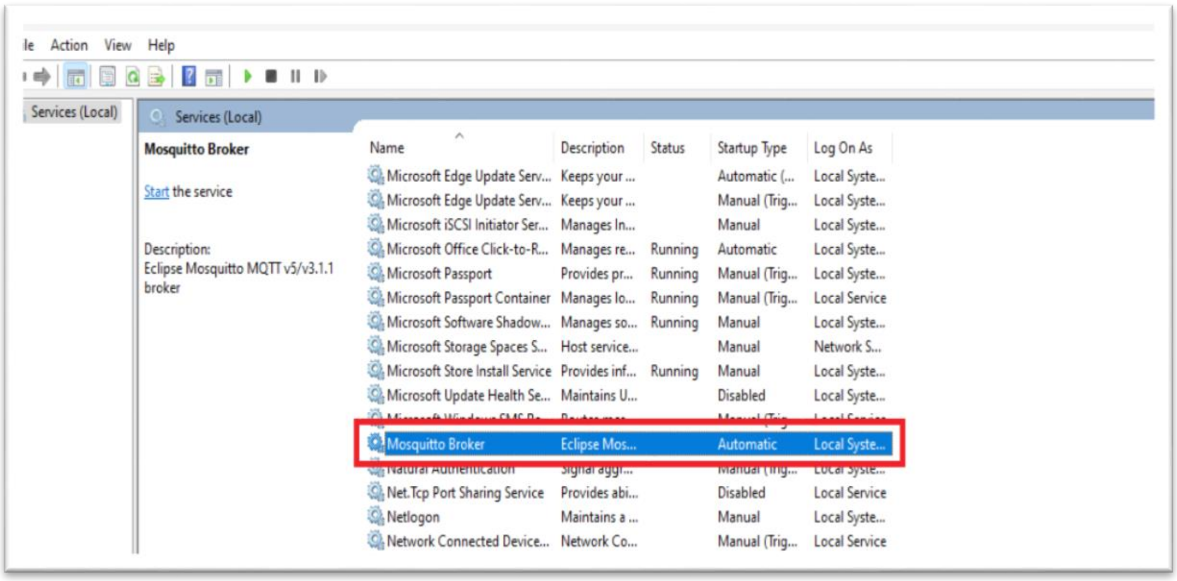
5. EXPERIMENTS :

We can use a free cloud MQTT broker or a locally installed MQTT broker to work with our designed application. The following steps need to be followed to work with a local broker.

Install Local Mosquitto Broker:

1) Download mosquitto from https://mosquitto.org/download/. We used the Windows 64-bit version. According to the operating system, we need to download the build version.

2) Install using the downloaded setup file.
3) From the command line, open services. Find the "Mosquitto Broker." Right-click on it and click on start. Figure 2 depicts the services inside the service window.



Mosquitto Broker

Start the service

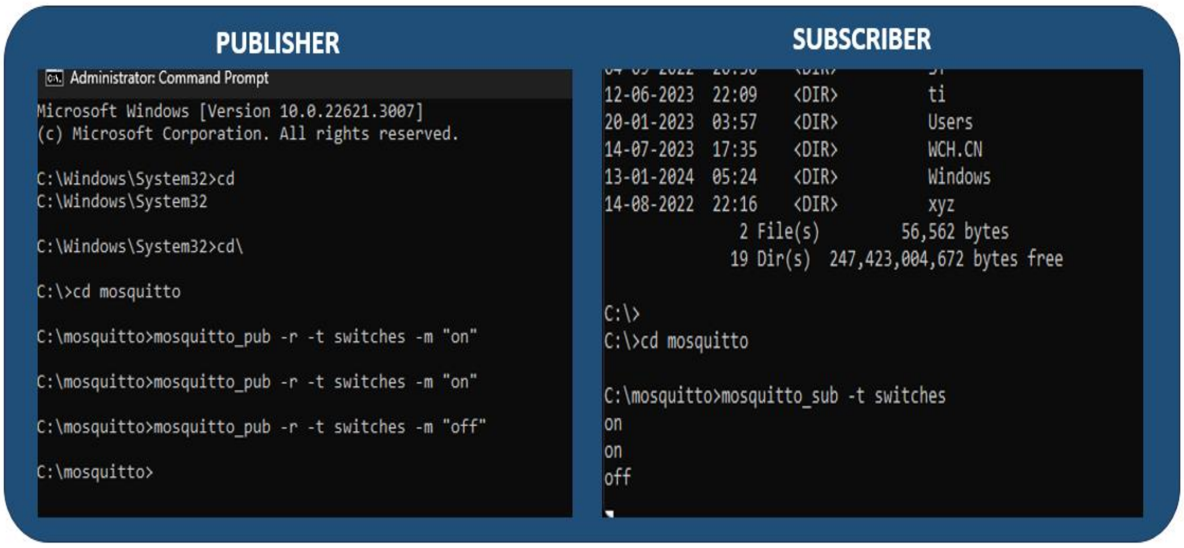
Description:
Eclipse Mosquitto MQTT v5/v3.1.1 broker

Name	Description	Status	Startup Type	Log On As
Microsoft Edge Update Serv...	Keeps your ...		Automatic (...	Local Syste...
Microsoft Edge Update Serv...	Keeps your ...		Manual (Trig...	Local Syste...
Microsoft iSCSI Initiator Ser...	Manages In...		Manual	Local Syste...
Microsoft Office Click-to-R...	Manages re...	Running	Automatic	Local Syste...
Microsoft Passport	Provides pr...	Running	Manual (Trig...	Local Syste...
Microsoft Passport Container	Manages lo...	Running	Manual (Trig...	Local Service
Microsoft Software Shadow...	Manages so...	Running	Manual	Local Syste...
Microsoft Storage Spaces S...	Host service...		Manual	Network S...
Microsoft Store Install Service	Provides inf...	Running	Manual	Local Syste...
Microsoft Update Health Se...	Maintains U...		Disabled	Local Syste...
Microsoft Windows SMS Ro...	Routes mes...		Manual (Trig...	Local Service
Mosquitto Broker	Eclipse Mos...		Automatic	Local Syste...
Natural Authentication	Signal aggr...		Manual (Trig...	Local Syste...
Net.Tcp Port Sharing Service	Provides abi...		Disabled	Local Service
Netlogon	Maintains a ...		Manual	Local Syste...
Network Connected Device...	Network Co...		Manual (Trig...	Local Service

Fig. 2: The mosquitto services inside the service window (Source: Author)

Test publisher and subscriber:

1) Open one command window.
2) Set prompt inside the mosquito broker install directory.
3) Type the command "**mosquitto_sub -t switches**." It will subscribe to the "switches" topic.
4) Now again, open another command window.
5) Type the command "**mosquitto_pub -r -t switches -m "on**"
6) Figure 3 depicts the publish and subscribe topic.



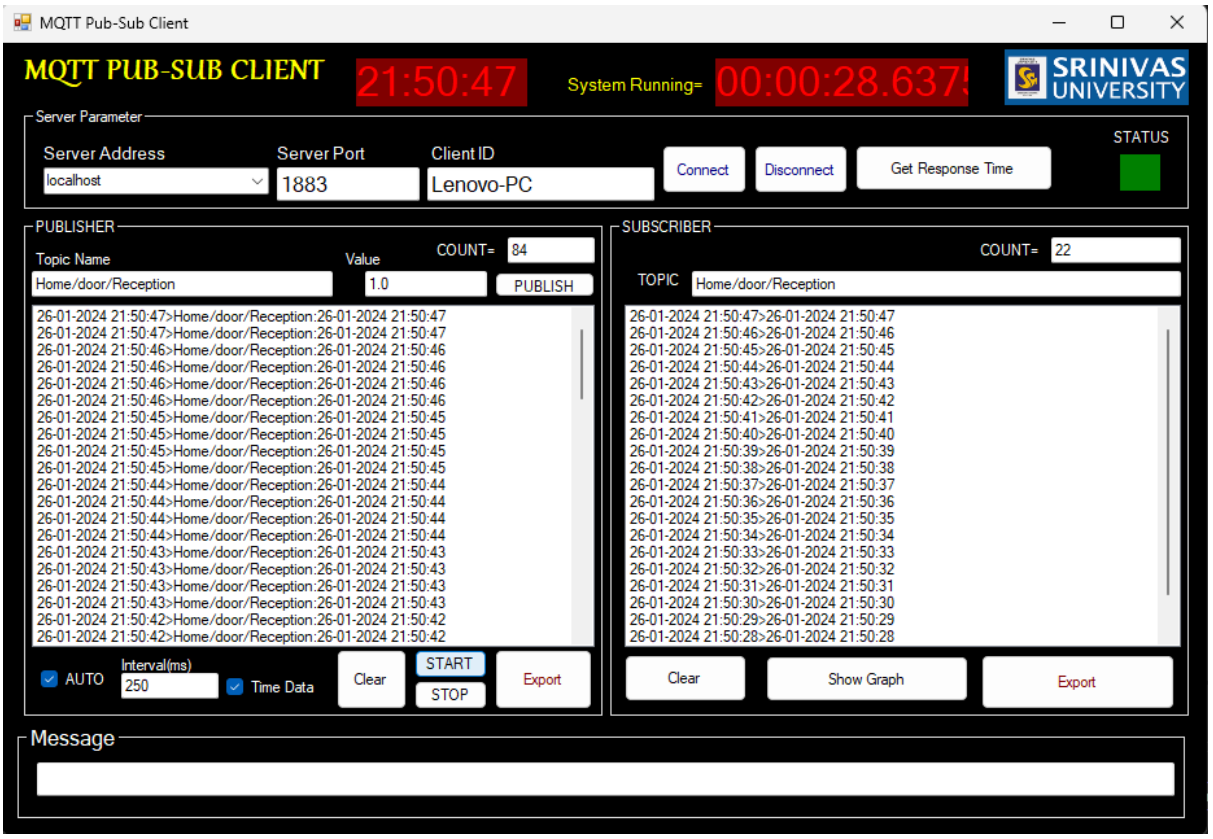
```
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd
C:\Windows\System32

C:\Windows\System32>cd\

C:\>cd mosquitto

C:\mosquitto>mosquitto_pub -r -t switches -m "on"

C:\mosquitto>mosquitto_pub -r -t switches -m "on"

C:\mosquitto>mosquitto_pub -r -t switches -m "off"

C:\mosquitto>
```

```
12-06-2023  22:09    <DIR>          ti
20-01-2023  03:57    <DIR>          Users
14-07-2023  17:35    <DIR>          WCH.CN
13-01-2024  05:24    <DIR>          Windows
14-08-2022  22:16    <DIR>          xyz
               2 File(s)         56,562 bytes
              19 Dir(s)  247,423,004,672 bytes free

C:\>
C:\>cd mosquitto

C:\mosquitto>mosquitto_sub -t switches
on
on
off
```

Fig. 3: The publish and subscribe topic (Source: Author)

MQTT Application Creation in C#:

1) we will create a Windows desktop application GUI in C# like Figure 4.



21:50:47 System Running= 00:00:28.637

Server Parameter
Server Address: localhost
Server Port: 1883
Client ID: Lenovo-PC

PUBLISHER
Topic Name: Home/door/Reception
Value: 1.0
COUNT= 84

```
26-01-2024 21:50:47>Home/door/Reception:26-01-2024 21:50:47
26-01-2024 21:50:47>Home/door/Reception:26-01-2024 21:50:47
26-01-2024 21:50:46>Home/door/Reception:26-01-2024 21:50:46
26-01-2024 21:50:46>Home/door/Reception:26-01-2024 21:50:46
26-01-2024 21:50:46>Home/door/Reception:26-01-2024 21:50:46
26-01-2024 21:50:46>Home/door/Reception:26-01-2024 21:50:46
26-01-2024 21:50:45>Home/door/Reception:26-01-2024 21:50:45
26-01-2024 21:50:45>Home/door/Reception:26-01-2024 21:50:45
26-01-2024 21:50:45>Home/door/Reception:26-01-2024 21:50:45
26-01-2024 21:50:45>Home/door/Reception:26-01-2024 21:50:45
26-01-2024 21:50:44>Home/door/Reception:26-01-2024 21:50:44
26-01-2024 21:50:44>Home/door/Reception:26-01-2024 21:50:44
26-01-2024 21:50:44>Home/door/Reception:26-01-2024 21:50:44
26-01-2024 21:50:44>Home/door/Reception:26-01-2024 21:50:44
26-01-2024 21:50:43>Home/door/Reception:26-01-2024 21:50:43
26-01-2024 21:50:43>Home/door/Reception:26-01-2024 21:50:43
26-01-2024 21:50:43>Home/door/Reception:26-01-2024 21:50:43
26-01-2024 21:50:43>Home/door/Reception:26-01-2024 21:50:43
26-01-2024 21:50:42>Home/door/Reception:26-01-2024 21:50:42
26-01-2024 21:50:42>Home/door/Reception:26-01-2024 21:50:42
```

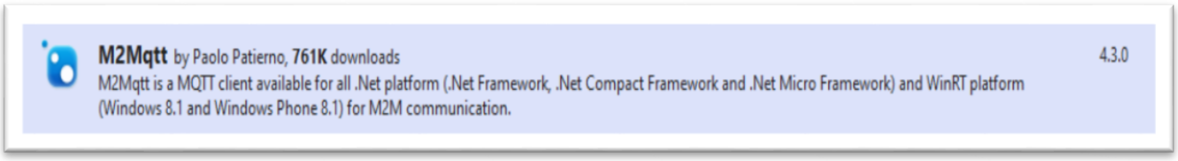
[✓] AUTO Interval(ms): 250 [✓] Time Data

SUBSCRIBER
COUNT= 22
TOPIC: Home/door/Reception

```
26-01-2024 21:50:47>26-01-2024 21:50:47
26-01-2024 21:50:46>26-01-2024 21:50:46
26-01-2024 21:50:45>26-01-2024 21:50:45
26-01-2024 21:50:44>26-01-2024 21:50:44
26-01-2024 21:50:43>26-01-2024 21:50:43
26-01-2024 21:50:42>26-01-2024 21:50:42
26-01-2024 21:50:41>26-01-2024 21:50:41
26-01-2024 21:50:40>26-01-2024 21:50:40
26-01-2024 21:50:39>26-01-2024 21:50:39
26-01-2024 21:50:38>26-01-2024 21:50:38
26-01-2024 21:50:37>26-01-2024 21:50:37
26-01-2024 21:50:36>26-01-2024 21:50:36
26-01-2024 21:50:35>26-01-2024 21:50:35
26-01-2024 21:50:34>26-01-2024 21:50:34
26-01-2024 21:50:33>26-01-2024 21:50:33
26-01-2024 21:50:32>26-01-2024 21:50:32
26-01-2024 21:50:31>26-01-2024 21:50:31
26-01-2024 21:50:30>26-01-2024 21:50:30
26-01-2024 21:50:29>26-01-2024 21:50:29
26-01-2024 21:50:28>26-01-2024 21:50:28
```

Fig. 4: The application interface (Source: Author).

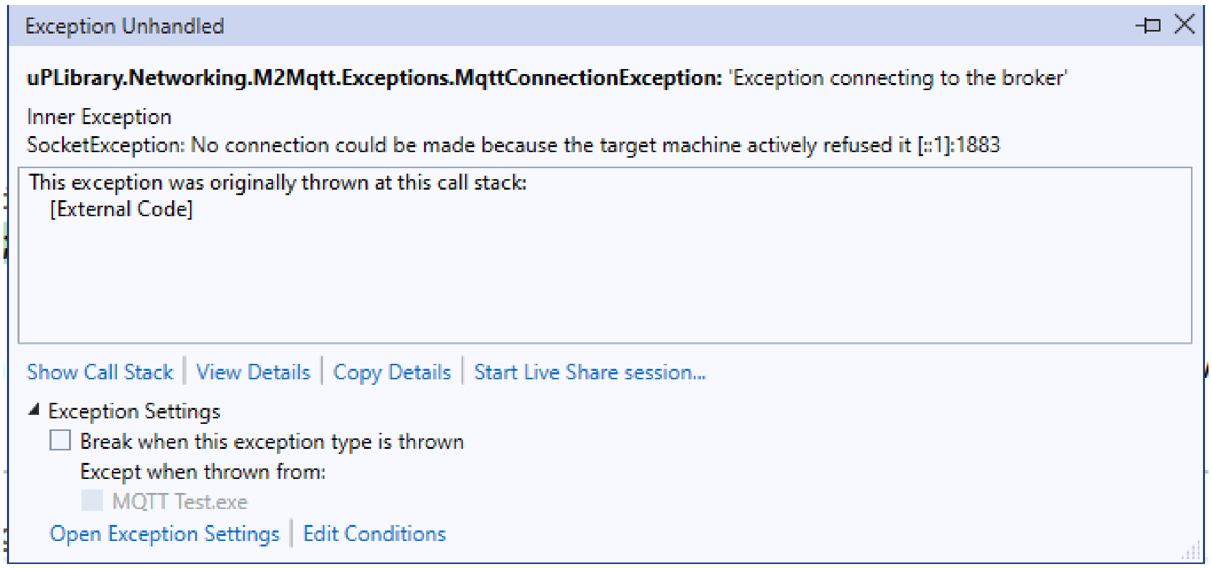
2) Download the project from https://github.com/sudipchakraborty/Let-Us-Build-a-MQTT-Pub-Sub-Client-In-C-sharp.git or an expert researcher can create the interface on their own.
3) Extract the project. Open in the visual studio. We used Visual Studio Community Edition 2022.
4) Using the Nuget package manager, uninstall the M2MQTT package and install it again due to the installation of missing components, which may not included inside the GitHub repository. Figure 5 depicts the MQTT package we used in our project.



M2Mqtt by Paolo Patierno, **761K** downloads 4.3.0
M2Mqtt is a MQTT client available for all .Net platform (.Net Framework, .Net Compact Framework and .Net Micro Framework) and WinRT platform (Windows 8.1 and Windows Phone 8.1) for M2M communication.

Fig. 5: The C# library for MQTT (Source: Author)

5) Build and run the project. If the project successfully builds and runs the application, it looks like it is in Figure 4. One common issue is that the MQTT broker needs to be running. Figure 6 depicts the error. In that scenario, we must run the services for the local host MQTT broker.
6) Select the server we want to set as an MQTT broker. The "localhost" is the default broker address when the application runs.
7) If we press the connect button, it will connect with the broker. If the connection is successful, the status shows green.
8) Then press the start button from the publisher side.


Exception Unhandled

uPLibrary.Networking.M2Mqtt.Exceptions.MqttConnectionException: 'Exception connecting to the broker'

Inner Exception
SocketException: No connection could be made because the target machine actively refused it [::1]:1883

This exception was originally thrown at this call stack:
[External Code]

Show Call Stack | View Details | Copy Details | Start Live Share session...
Exception Settings
Break when this exception type is thrown
Except when thrown from:
MQTT Test.exe
Open Exception Settings | Edit Conditions

Fig. 6: The error on not running the MQTT broker

9) The published data should be visible inside the list box.
10) If the name of the subscribe is the same topic, the published topic will be displayed inside the subscribe list box.
11) Press the “disconnect” button to disconnect from the broker. Change the broker and experiment.
12) The GUI controls are self-explanatory.
13) Some control just added. It is kept for the researcher to add code as their project needs.
14) The “Get response time” button can be implemented to study the broker's performance. It is the time difference between publishing and subscribing to the same topic to the same application.
15) The “Show Graph” button can show the subscriber topic in the graph form to better understand the sensor data. The C# experienced researcher can do it quickly.

6. RESULTS & DISCUSSIONS :

The data published interval of more than or equal to 500 milliseconds is good. We didn’t observe noticeable changes if we set the publish interval to less than 500 milliseconds. We followed the receive frequency of one second or more on the subscriber side. We also observed that the received topic interval is not always steady where the publishing interval is constant. We tested several MQTT data propagations over the internet. The final result is not much different compared to other brokers. The MQTT is a promising communication channel. But we must consider many factors before deployment, like server selection, broker or server bandwidth, network speed, etc. Overall, the good thing is that it is easy to implement. From the programmer's perspective, it is a fast and easy prototyping communication channel to integrate into their project.

7. ANALYSIS / COMPARISON OF RESULTS :

We tested practical sensor data transfer between two nodes. We tested locally. For the cloud server, we used “broker.hivemq.com,” which is better than other servers. The local broker or server has the least response time. However, a local server is not always feasible. We have an option for that. Using the local server, we can utilize the tunnel to expose the other node over the internet. But overall, data propagation depends on the data traffic present in the network. We might suggest not using it to trigger critical devices like furnaces or life support devices because there are no guarantees that data will reach the subscribed client node within the expected interval. It can store the log or sensor data in the server or display parameters.

8. SUGGESTIONS / RECOMMENDATIONS :

We provide suggestions to improve the software or the knowledge base for the research work.

- A practical resource on MQTT using C# is: https://www.youtube.com/watch?v=1IJAfiBkp1E
- To expose the local MQTT broker to the net: https://www.youtube.com/watch?v=HU04XApJxvk

- ➢ To get a list of free MQTT brokers, https://mntolia.com/10-free-public-private-mqtt-brokers-for-testing-prototyping/
- ➢ We didn't include the exception handle to keep the entire project easy to understand for the new researcher. The researcher needs to add exceptions for the flawless running of the application.
- ➢ Export features should be implemented to export the incoming subscribe topic, which is displayed in the list box. It should facilitate export in various formats.
- ➢ We added a "Show Graph" button. The button's function is to show the incoming topic as a graph. It will take the listbox data as input, parse it, and feed it to the graph control. Then, it will be displayed inside the graph. We kept it to be implemented by the researcher if needed.
- ➢ We added one button called "Get Response Time." The button's function is to measure the response time of the broker, which is selected from the combo box.

9. CONCLUSION :

To work with the MQTT protocol, we need an excellent MQTT pub-sub client application to test with various brokers and subscribe or publish topics. But most of the available free software, either time limitations or some features, are available only in the premium version. Some research projects need help to afford to purchase tools. Also, if we use the paid version, some features must be made available, which is a common requirement of our researcher. So here we provide a way to create our own MQTT debugger. We offer some noticeable research in this field, and the methodology demonstrates the process flow of the MQTT communication channel. In the experiment, we provide a step-by-step guide to building the application in our lab. We added some unique features that are not available in other applications. We added some features through buttons, which we finally kept to be implemented by the researcher as needed.

10. ACKNOWLEDGEMENT :

- I express my heartfelt gratitude to Dr. P. S. Aithal, my esteemed research supervisor, for his invaluable guidance, supervision, encouragement, and unwavering support throughout this research endeavour.
- Srinivas University deserves my thanks for generously providing the essential resources, facilities, and assistance that significantly contributed to the success of this study.
- My sincere appreciation goes to my family for their continuous support, understanding, and motivation during the research journey.
- Special thanks to my loving son, who consistently energized me when I faced challenges, and my daughter, who remained a source of inspiration during moments of boredom.
- I acknowledge the collective help and encouragement from various remarkable individuals and institutions, without whom this work would not have been possible.

11. LIMITATIONS :

We provide an excellent tool for the MQTT research work. Despite that, there are a couple of limitations we listed below:

- ❖ For security or encryption, MQTT needs to be implemented into the communication part.
- ❖ User input validation needs to be implemented.
- ❖ There are several brokers available for IoT communication. We googled and added the server combo box. Except for broker.hivemq.com, we didn't get a good performance and could not communicate easily. The researcher can investigate with other available free brokers or servers.

REFERENCES :

[1] Velinov, A., Mileva, A., Wendzel, S., & Mazurczyk, W. (2019). Covert channels in the MQTT-based Internet of Things. *IEEE Access, 7*(1), 161899-161915. Google Scholar↗

[2] Longo, E., Redondi, A. E. C., Cesana, M., & Manzoni, P. (2022). BORDER: A benchmarking framework for distributed MQTT brokers. *IEEE Internet of Things Journal, 9*(18), 17728-17740. Google Scholar↗

[3] Hästbacka, D., Kannisto, P., & Kätkytniemi, A. (2022, October). Interoperability of OPC UA PubSub with existing message broker integration architectures. In *IECON 2022–48th Annual Conference of the IEEE Industrial Electronics Society* (pp. 1-6). IEEE. Google Scholar↗

[4] Manzoni, P., Maniezzo, V., & Boschetti, M. A. (2022). Modeling Distributed MQTT Systems Using Multicommodity Flow Analysis. *Electronics*, *11*(9), 1498. Google Scholar↗

[5] Walker, M., von Arnim, C., Neubauer, M., Lechler, A., Riedel, O., & Verl, A. (2023, April). Reflection-based Prototyping Framework for OPC UA Servers for Companion Specifications. In 2023 IEEE International Conference on Industrial Technology (ICIT) (pp. 1-6). IEEE. Google Scholar↗

[6] Gupta, V., Khera, S., & Turk, N. (2021). MQTT protocol employing IOT based home safety system with ABE encryption. *Multimedia Tools and Applications*, *80*(2), 2931-2949. Google Scholar↗

[7] Gültunca, C., & Zaim, A. H. (2018). Examination and comparison of the communication protocols on the application layer in iot. *İstanbul Ticaret Üniversitesi Fen Bilimleri Dergisi*, *17*(33), 41-50. Google Scholar↗

[8] Al-Masri, E., Kalyanam, K. R., Batts, J., Kim, J., Singh, S., Vo, T., & Yan, C. (2020). Investigating messaging protocols for the Internet of Things (IoT). *IEEE Access*, *8*, 94880-94911. Google Scholar↗

[9] Chang, H., Hao, F., Kodialam, M., Lakshman, T. V., Mukherjee, S., & Varvello, M. (2023). Towards network-assisted publish–subscribe over wide area networks. *Computer Networks*, *231*(1), 109702. Google Scholar↗

[10] Roffia, L., Morandi, F., Kiljander, J., D'Elia, A., Vergari, F., Viola, F., ... & Cinotti, T. S. (2016). A semantic publish-subscribe architecture for the Internet of Things. *IEEE Internet of Things Journal*, *3*(6), 1274-1296. Google Scholar↗

[11] Chakraborty, S., & Aithal, P. S., (2022). A Practical Approach to GIT Using Bitbucket, GitHub and SourceTree. International Journal of Applied Engineering and Management Letters (IJAEML), 6(2), 254-263. DOI: https://doi.org/10.5281/zenodo.7262771

[12] Chakraborty, S., & Aithal, P. S., (2022). How to make IoT in C# using Sinric Pro. International Journal of Case Studies in Business, IT, and Education (IJCSBE), 6(2), 523- 530. DOI: https://doi.org/10.5281/zenodo.7335167

[13] Chakraborty, S., & Aithal, P. S., (2022). Virtual IoT Device in C# WPF Using Sinric Pro. International Journal of Applied Engineering and Management Letters (IJAEML), 6(2), 307-313. DOI: https://doi.org/10.5281/zenodo.7473766

[14] Chakraborty, S., & Aithal, P. S., (2023). MVVM Demonstration Using C# WPF. International Journal of Applied Engineering and Management Letters (IJAEML), 7(1), 1- 14. DOI: https://doi.org/10.5281/zenodo.7538711

[15] Chakraborty, S., & Aithal, P. S., (2023). Let Us Create An IoT Inside the AWS Cloud. International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7(1), 211- 219. DOI: https://doi.org/10.5281/zenodo.7726980

[16] Chakraborty, S., & Aithal, P. S., (2023). Let Us Create a Physical IoT Device Using AWS and ESP Module. International Journal of Management, Technology, and Social Sciences (IJMTS), 8(1), 224-233. DOI: https://doi.org/10.5281/zenodo.7779097

[17] Chakraborty, S., & Aithal, P. S., (2023). Let Us Create Multiple IoT Device Controller Using AWS, ESP32 And C#. International Journal of Applied Engineering and Management Letters (IJAEML), 7(2), 27-34. DOI: https://doi.org/10.5281/zenodo.7857660

[18] Chakraborty, S., & Aithal, P. S., (2023). Let Us Create Our Desktop IoT Soft-Switchboard Using AWS, ESP32 and C#. International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7(3), 185-193. DOI: https://doi.org/10.5281/zenodo.8234036

[19] Chakraborty, S. & Aithal, P. S. (2023). Let Us Create an Alexa Skill for Our IoT Device Inside the AWS Cloud. International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7(2), 214-225. DOI: https://doi.org/10.5281/zenodo.7940237

[20] Chakraborty, S., & Aithal, P. S. (2023). Let Us Create A Lambda Function for Our IoT Device In The AWS Cloud Using C#. International Journal of Management, Technology, and Social Sciences (IJMTS), 8(2), 145-155. DOI: https://doi.org/10.5281/zenodo.7995727

[21] Chakraborty, S., & Aithal, P. S. (2023). Modbus Data Provider for Automation Researcher Using C#. International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7(3), 1-7. DOI: https://doi.org/10.5281/zenodo.8162680

[22] Chakraborty, S., & Aithal, P. S. (2023). IoT-Based Industrial Debug Message Display Using AWS, ESP8266 And C#. International Journal of Management, Technology, and Social Sciences (IJMTS), 8(3), 249-255. DOI: https://doi.org/10.5281/zenodo.8250418

[23] Chakraborty, S., & Aithal, P. S. (2023). IoT-Based Switch Board for Kids Using ESP Module And AWS. International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7(3), 248-254. DOI: https://doi.org/10.5281/zenodo.8285219

[24] Chakraborty, S., & Aithal, P. S. (2023). Let Us Create an Alexa-Enabled IoT Device Using C#, AWS Lambda and ESP Module. International Journal of Management, Technology, and Social Sciences (IJMTS), 8(3), 256-261. DOI: https://doi.org/10.5281/zenodo.8260291

[25] Chakraborty, S., & Aithal, P. S. (2023). Alexa Enabled IoT Device Simulation Using C# And AWS Lambda. International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7(3), 359-368. DOI: https://doi.org/10.5281/zenodo.8329375

[26] Chakraborty, S., & Aithal, P. S. (2023). CRUD Operation on WordPress Database Using C# SQL Client. International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7(4), 138-149. DOI: https://doi.org/10.5281/zenodo.10162719

[27] Chakraborty, S., & Aithal, P. S., (2023). CRUD Operation On WordPress Database Using C# And REST API. International Journal of Applied Engineering and Management Letters (IJAEML), 7(4), 130-138. DOI: https://doi.org/10.5281/zenodo.10197134

[28] Chakraborty, S., & Aithal, P. S., (2023). CRUD Operation on WordPress Posts From C# over REST API. International Journal of Management, Technology, and Social Sciences (IJMTS), 8(4), 223-231. DOI: https://doi.org/10.5281/zenodo.10264407

[29] Chakraborty, S. & Aithal, P. S. (2023). CRUD Operation On WordPress Custom Post Type (CPT) From C# Over REST API. International Journal of Case Studies in Business, IT, and Education (IJCSBE), 7(4), 323-331. DOI: https://doi.org/10.5281/zenodo.10408545

[30] Chakraborty, S. & Aithal, P. S. (2023). Let Us Build a WordPress Custom Post Type (CPT). International Journal of Applied Engineering and Management Letters (IJAEML), 7(4), 259-266. DOI: https://doi.org/10.5281/zenodo.10440842

****