

MQTT Uses and Benefits over Ajax

(Project Paper)

Stanley Yu

Dr. Bantel Winfried

Project in Computer Science

February 13, 2023

Table of Contents

Introduction	3
What is Mosquitto (MQTT)?	3
How MQTT Works	4
Publish and Subscribe	4
Broker	4
Client	5
Topics	5
Plugins	6
Project Objectives	6
Implementing MQTT in a chat room website	6
Project Implementation	8
Programming Languages Used	8
Libraries and Tools Used	8
Challenges and Solutions	8
MQTT Compared to AJAX	12
What is AJAX?	
How AJAX Works	12
Why MQTT is better than AJAX for web browsers	13
Things to be Improved and Retrospective	13
What Could be Improved?	13
Lessons Learned	14
References	15

Introduction

What is Mosquitto (MQTT)?

MQTT, which stands for Message Queuing Telemetry Transport, is a lightweight publish-subscribe messaging protocol that is designed to be used in low-bandwidth, high-latency or unreliable networks. It was originally created by Dr. Andy Stanford-Clark and Arlen Nipper from IBM in 1999 with the purpose of monitoring devices used in the oil and gas industry to send their data to remote servers (OASIS Standard, 2013). In the present day, MQTT is a widely standard used for IoT (Internet of Things) and M2M communication because of its lightweight design, efficient data transfer and security features (Eberspächer, 2017).



Figure 1.1 - MQTT Logo

The architecture of MQTT is based on the TCP/IP protocol stack, which makes it a reliable, secure and fast data transfer protocol. This makes MQTT suitable for industrial and commercial environments where data integrity and reliability are crucial (Eberspächer, 2017). In addition, Mosquitto provides security features such as encrypted communication and access control that help to ensure that sensitive data is protected from unauthorized access.

Overall, MQTT is a widely used messaging protocol that has become the standard for IoT and M2M communication because of its lightweight design, efficient data transfer and security features (Eberspächer, 2017).

How MQTT Works

Publish and Subscribe

The communication component for MQTT works based on a publish and subscribe system in devices subscribed to the publisher's topic receive a message. This communication model contains three components: a publisher, a broker and a subscriber.

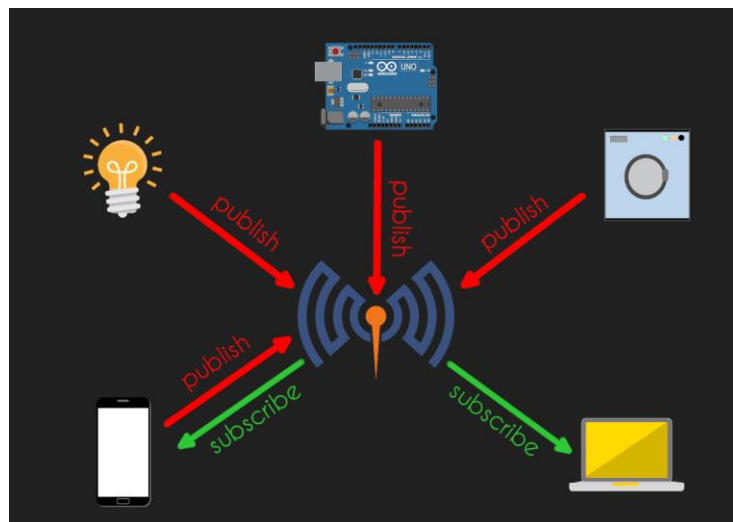


Figure 1.2 Publish and Subscriber Model

This exchange is depicted by Figure 1.2 above in which the phone and computer devices are receiving messages from the IoT devices who are the publishers. This shows that devices can either publish data to a topic or subscribe to a topic in order to receive data from other devices.

Broker

The broker is responsible for receiving messages from the publisher and distributing them to subscribers who subscribed to a topic from the broker. When the broker receives a message, it also has to filter the messages and routing messages to the correct recipient.

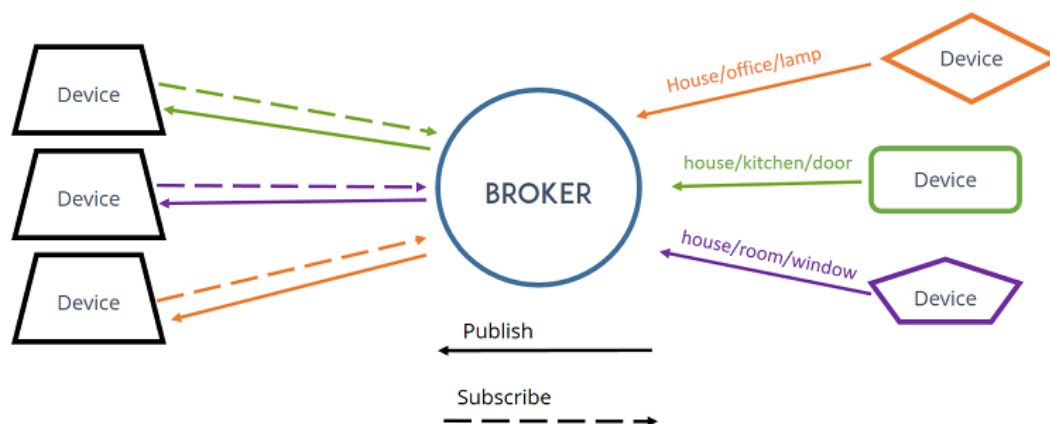


Figure 1.3 – Broker Component in MQTT

The broker can be depicted in Figure 1.3 above in which the broker acts as the main hub for communication as it receives and reroutes messages between devices.

Client

From Figure 1.2 and Figure 1.3, the devices mentioned are formally named clients in MQTT. Clients are devices that communicate with a broker to publish or subscribe to messages (OASIS Standard, 2013). An MQTT client could also be publishers, subscribers or both.

Topics

In MQTT, topics are a form of addressing that allows MQTT clients to share information. Topics are structured in a hierarchy similar to folders and files in a file system using “/” as a delimiter. Topic names are: case sensitive, use UTF-8 strings, and must consist of at least one character to be valid. If there are no topics posted by the clients, then “\$SYS” is used as the default topic.

An example for a topic would be denoted as:

- /
- /tower
- tower/room
- tower/room/light-switch

Plugins

Plugins for C++ are implemented as dynamic shared objects or shared libraries. For the MQTT library in C, plugins are additional software components that can be added to a MQTT broker in order to implement additional functionalities.

Additional features that could be included in a MQTT plugin would be adding data persistence so that the messages are stored even if the broker is restarted. Another plugin could be adding authentication and authorization capabilities with only authorized clients being able to access the broker.

Project Objectives

Implementing MQTT in a chat room website

The main objective for this project was implementing MQTT in a chat room website using a plugin as a shared library object to the latest version of the Mosquitto library. Before I could start implementing the application, I was tasked to research MQTT and answer these questions:

- With some return codes the server disconnects the client. Which return codes are this?

Answer:

`MQTT_RC_PROTOCOL_ERROR` (1) - Indicates a protocol error and the connection should be closed.

`MQTT_RC_IMPLEMENTATION_SPECIFIC_ERROR` (2) - Indicates an implementation-specific error and the connection should be closed.

`MQTT_RC_UNSUPPORTED_PROTOCOL_VERSION` (3) - Indicates that the client is using an unsupported protocol version and the connection should be closed.

MQTT_RC_CLIENT_IDENTIFIER_NOT_VALID (4) - Indicates that the client identifier is not valid and the connection should be closed.

MQTT_RC_BAD_USER_NAME_OR_PASSWORD (5) - Indicates that the username or password is not valid and the connection should be closed.

MQTT_RC_NOT_AUTHORIZED (6) - Indicates that the client is not authorized and the connection should be closed.

MQTT_RC_SERVER_UNAVAILABLE (8) - Indicates that the server is temporarily unavailable and the connection should be closed.

MQTT_RC_SERVER_BUSY (9) - Indicates that the server is busy and the connection should be closed.

MQTT_RC_BANNED (10) - Indicates that the client is banned and the connection should be closed.

- What's the reason for the Message-Callback? Why couldn't everything be done using ACL-Callback?

Answer:

The purpose for Message-Callback is to handle incoming messages by triggering whenever a message is received on a subscribed topic. This allows the processing of incoming messages and makes it possible to perform actions such as updating the user interface or sending notifications.

While ACL (Access Control List) Callback could be used to handle everything, ACL-Callback is typically used for security based tasks such as authentication and authorization while Message-Callback is focused on handling incoming messages. It could be easier to maintain code if these responsibilities are split into separate callbacks instead of using ACL-Callback for everything.

These questions provide context on why using ACL-Callback could be useful in developing a chat application instead of using Message-Callback. Overall, the purpose of this project is to document how MQTT works and to develop an chat application using an MQTT plugin.

Project Implementation

Programming Languages Used

The programming languages used for this project are:

- C++
- Python
- JavaScript
- HTML

Libraries and Tools Used

The libraries used for this project are:

- mosquitto.h
- mosquitto_plugin.h
- mosquitto_broker.h
- string.h
- stdio.h

The tools used for this project are:

- Ubuntu
- Firefox
- Gedit

Challenges and Solutions

Throughout the development process, there were several challenges that I encountered and found the solution to:

Configuring the VM:

Challenge:

It was difficult getting the provided virtual machine with MQTT preinstalled to work with VirtualBox.

Port 1	Port 2	Port 3	Port 4
<input checked="" type="checkbox"/> Enable Serial Port			
Port Number: COM1		IRQ: 4	I/O Port: 0x3F8
Port Mode: Raw File			
<input checked="" type="checkbox"/> Connect to existing pipe/socket			
Path/Address:			

Figure 2.1 – VirtualBox VM Configuration

As shown in Figure 2.1 above, there were issues with configuring the VM since it was missing a Path/Address to the Raw File.

Solution:

The solution to this issue was installing Ubuntu using WSL (Windows Subsystem for Linux) which allowed me to run a Linux environment on a Windows system.

```

stan@DESKTOP-F05IL6M: ~/MQTT$ ls
api mosquitto-1.5.3 mosquitto-1.5.3.tar.gz mqtt-fetch.js paho-python plugin subscriber_ex
stan@DESKTOP-F05IL6M: ~/MQTT$ cd mqtt-fetch.js/
stan@DESKTOP-F05IL6M: ~/MQTT/mqtt-fetch.js$ ls
LICENSE README.md examples ibs.png src
stan@DESKTOP-F05IL6M: ~/MQTT/mqtt-fetch.js$ cd examples/
stan@DESKTOP-F05IL6M: ~/MQTT/mqtt-fetch.js/examples$ ls
chat-complex chat-simple square
stan@DESKTOP-F05IL6M: ~/MQTT/mqtt-fetch.js/examples$ cd chat-simple/
stan@DESKTOP-F05IL6M: ~/MQTT/mqtt-fetch.js/examples/chat-simple$ ls
chat-simple-backend.c chat-simple-backend.so lern.c lib main.c main.o mqtt-chat.html
supervisord.conf
stan@DESKTOP-F05IL6M: ~/MQTT/mqtt-fetch.js/examples/chat-simple$
stan@DESKTOP-F05IL6M: ~/MQTT/mqtt-fetch.js/examples/chat-simple$
stan@DESKTOP-F05IL6M: ~/MQTT/mqtt-fetch.js/examples/chat-simple$ mosquitto -v
1675848440: mosquitto version 2.0.15 starting
1675848440: Using default config.
1675848440: Starting in local only mode. Connections will only be possible from clients running on t
his machine.
1675848440: Create a configuration file which defines a listener to allow remote access.
1675848440: For more details see https://mosquitto.org/documentation/authentication-methods/
1675848440: Opening ipv4 listen socket on port 1883.
1675848440: Opening ipv6 listen socket on port 1883.
1675848440: mosquitto version 2.0.15 running

```

Figure 2.2 – Ubuntu WSL With Mosquitto Running

By using Ubuntu on WSL, I was able to run the latest version of MQTT along with compiling and testing the chat application for the project.

Challenge:

By switching to WSL Ubuntu I had a different issue which was being unable to load a graphical interface.

```
stan@DESKTOP-F05IL6M:~/MQTT/mqtt-fetch.js/examples/chat-simple$ firefox mqtt-chat.html
Error: no DISPLAY environment variable specified
```

Figure 2.3 – Error Stating “no DISPLAY environment variable specified”

Solution:

I was able to solve this issue by installing a GUI package using the “sudo apt full-upgrade” command and by changing the configuration file to enable *systemd*.

Challenge:

While debugging the website, I frequently encountered errors with the API stating:

(Cannot send a message with user profiles)

(“Firefox can’t establish a connection to the server at ws://test.mosquitto.org:8080/”)

(“Uncaught (in promise) TypeError: c.chats is undefined”)

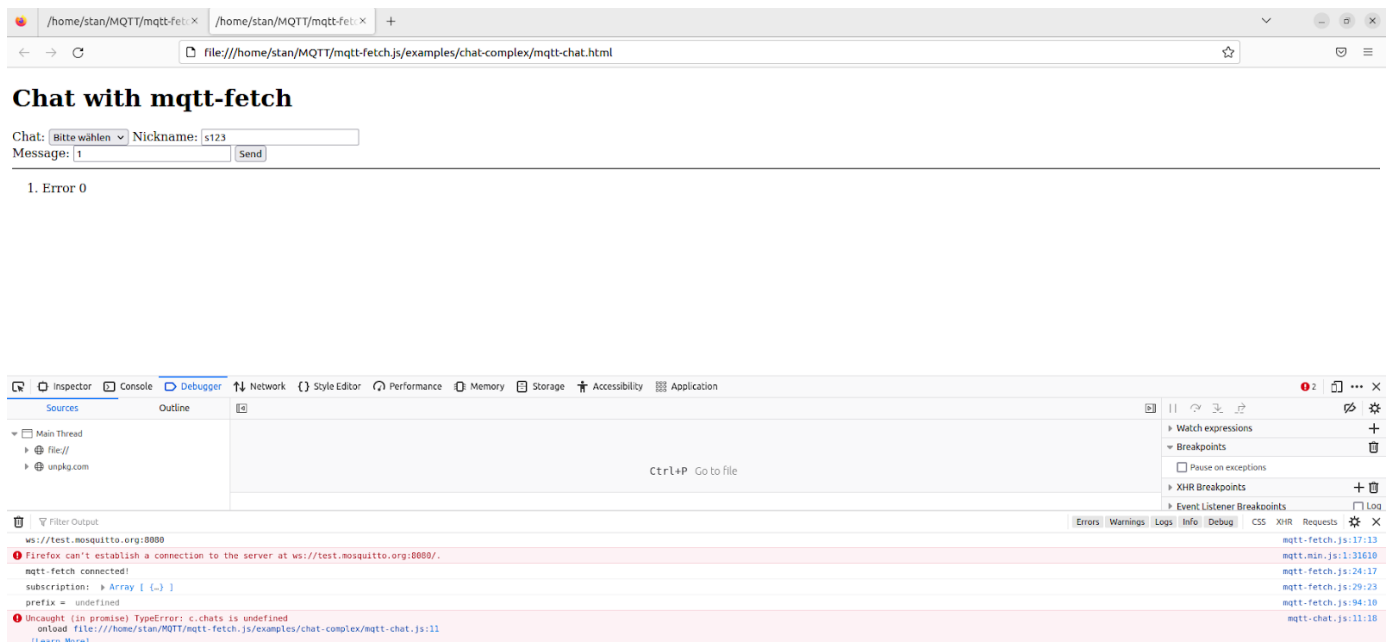
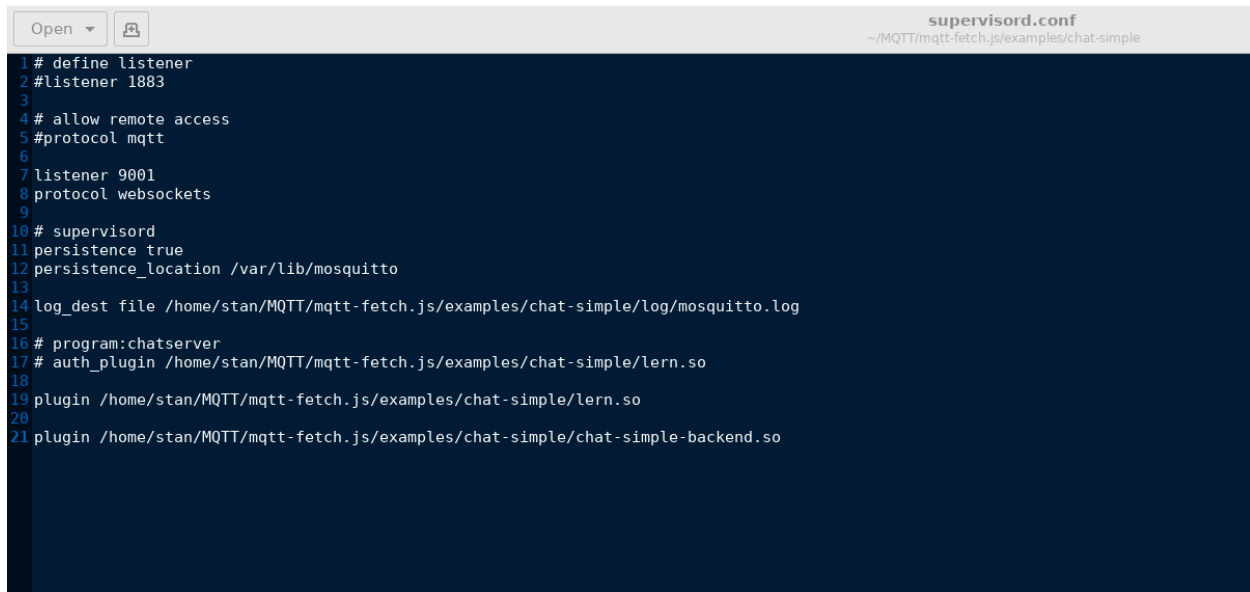


Figure 2.4 – MQTT-Chat Application Errors

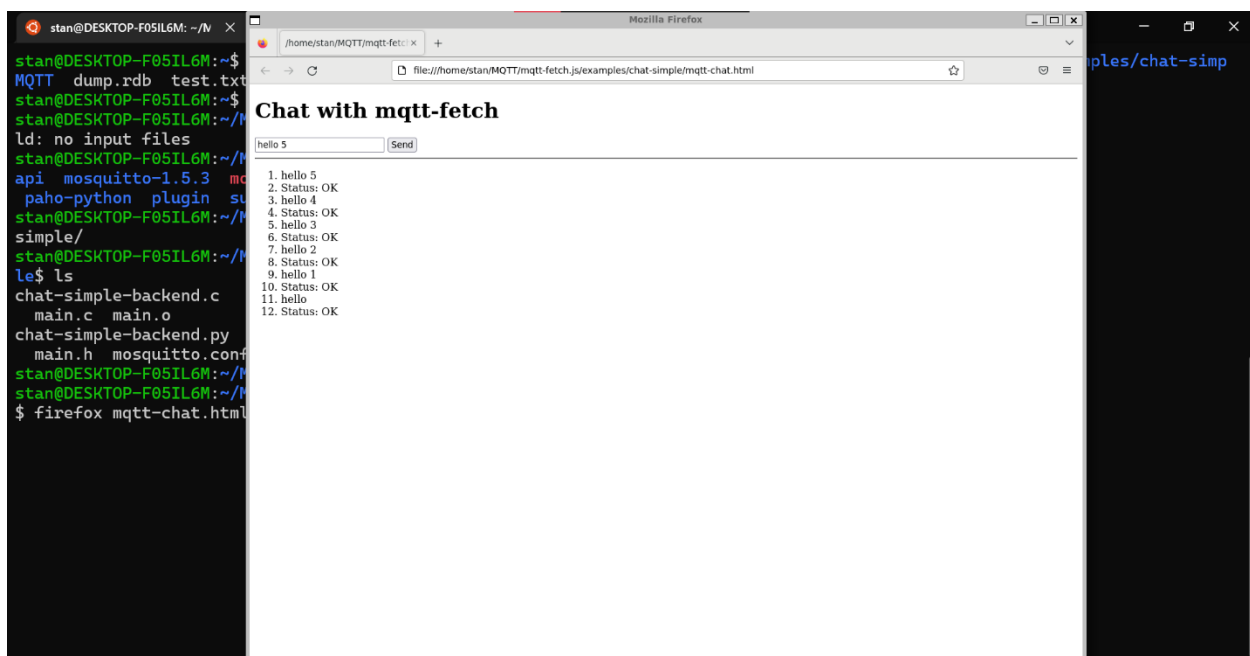
Solution:

The solution to this issue was to rewrite the MQTT configuration file, *supervisord.conf*, to correctly compile the plugins with the chat application website shown in Figure 2.6 where messages are correctly posted. I also modified the file to log errors in the directory I specified as shown in Figure 2.5 below.

A screenshot of a text editor window showing the configuration file `supervisord.conf`. The file path is `~/MQTT/mqtt-fetch.js/examples/chat-simple`. The configuration includes settings for a listener on port 1883, allowing remote access, and a websocket listener on port 9001. It also defines a supervisor named `chatserver` with persistence enabled, logging to `/var/lib/mosquitto`, and two plugins: `lern.so` and `chat-simple-backend.so`.

```
1 # define listener
2 #listener 1883
3
4 # allow remote access
5 #protocol mqtt
6
7 listener 9001
8 protocol websockets
9
10 # supervisord
11 persistence true
12 persistence_location /var/lib/mosquitto
13
14 log_dest file /home/stan/MQTT/mqtt-fetch.js/examples/chat-simple/log/mosquitto.log
15
16 # program:chatserver
17 # auth_plugin /home/stan/MQTT/mqtt-fetch.js/examples/chat-simple/lern.so
18
19 plugin /home/stan/MQTT/mqtt-fetch.js/examples/chat-simple/lern.so
20
21 plugin /home/stan/MQTT/mqtt-fetch.js/examples/chat-simple/chat-simple-backend.so
```

Figure 2.5 – MQTT Configuration File

A screenshot showing a terminal window and a web browser. The terminal window shows the user `stan` at a desktop with IP `F051L6M`. They run `MQTT dump.rdb test.txt`, `ld: no input files`, and `api mosquitto-1.5.3`. They also run `ls` in the `chat-simple-backend.c` directory, showing files `main.c`, `main.o`, `chat-simple-backend.py`, and `main.h`. Finally, they run `firefox mqtt-chat.html`. The web browser window shows the `mqtt-chat.html` file, which displays a chat interface titled "Chat with mqtt-fetch". The chat interface has a text input field with "hello 5" and a "Send" button. Below the input field, a list of chat messages is displayed, including "hello 5", "Status: OK", "hello 4", "Status: OK", "hello 3", "Status: OK", "hello 2", "Status: OK", "hello 1", "Status: OK", "hello", and "Status: OK".

```
stan@DESKTOP-F051L6M: ~/$
MQTT dump.rdb test.txt
stan@DESKTOP-F051L6M: ~/$
ld: no input files
stan@DESKTOP-F051L6M: ~/$
api mosquitto-1.5.3
paho-python plugin su
stan@DESKTOP-F051L6M: ~/$
simple/
stan@DESKTOP-F051L6M: ~/$
le$ ls
chat-simple-backend.c
main.c main.o
chat-simple-backend.py
main.h mosquitto.conf
stan@DESKTOP-F051L6M: ~/$
stan@DESKTOP-F051L6M: ~/$
$ firefox mqtt-chat.html
```

Figure 2.6 – MQTT Chat Application Working

MQTT Compared to AJAX

What is AJAX?

AJAX (Asynchronous JavaScript and XML) is a web development technique that allows developers to create fast and dynamic web pages. AJAX can allow developers to read data from a web browser after a web page has been loaded, update a web page without reloading the page and send data to a web server in the background.

According to W3Schools (2021), AJAX is a combination of technologies that work together to create asynchronous applications. It uses JavaScript to send and receive server data without having to reload the entire page which results in faster and more dynamic web pages.

How AJAX Works

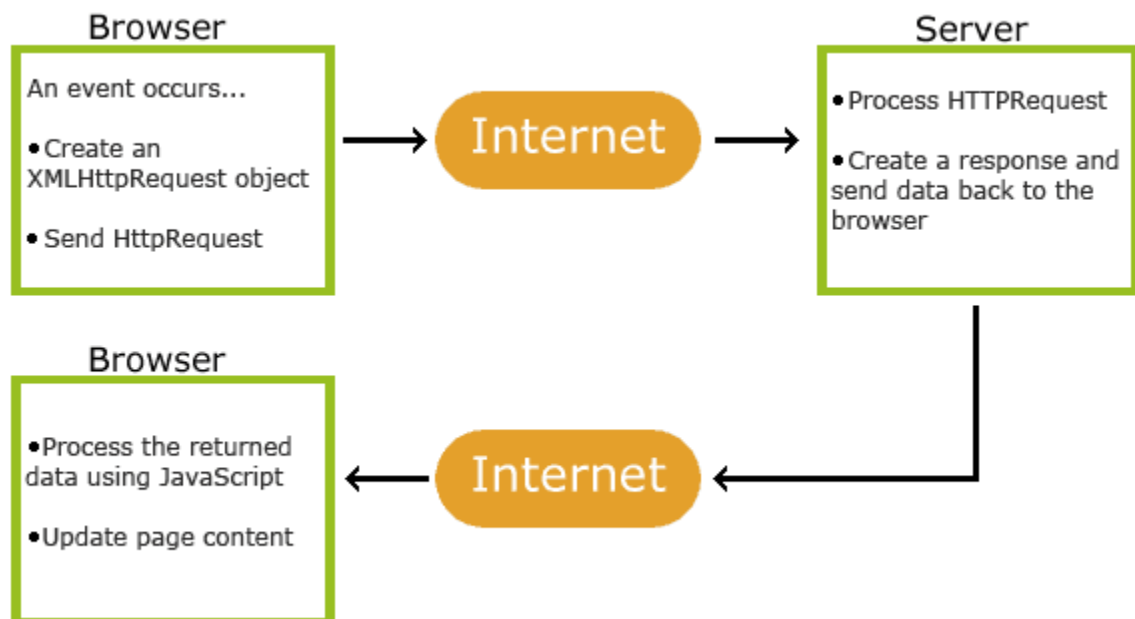


Figure 3.1 – AJAX Communication

As shown in Figure 3.1, AJAX communication works by going through a series of steps between a Browser and a Server using JavaScript and XMLHttpRequest objects (W3Schools 2021):

1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

Why MQTT is better than AJAX for web browsers

While both MQTT and AJAX have their benefits for developing web pages, MQTT is better suited for use in web browsers for certain circumstances.

One of the reasons for this is MQTT's efficient use of bandwidth since MQTT is a lightweight protocol. In MQTT, the server would only send data to clients that are subscribed to receive it, which reduced the amount of unnecessary data transmitted over the network. On the other hand, AJAX sends a request to the server for every interaction, which can result in a large amount of data sent (OASIS, 2021).

Another reason why MQTT is better than AJAX is that MQTT is designed to be a secure protocol with authentication and encryption built-in. This makes it useful for web applications where security is a concern, such as banking or government websites (Eclipse Foundation, 2021).

Things to be Improved and Retrospective

What Could be Improved?

After finishing the project, there are some components I would improve in the future with more time. The main component I would focus on improving would be creating an account system for the simple chat system. I would do this by adding the ACL-callback function to the shared library plugin and adjusting the JavaScript code to save credentials.

Lessons Learned

Throughout this project, I have learned a lot about how devices use the MQTT protocol to communicate with each other. By developing the chat application, I learned how to compile and use a shared library along with learning how run a Mosquitto Broker with a custom configuration file. The lessons I have learned are valuable in the future for developing secure web applications as I have the knowledge to implement MQTT into future web pages for secure authentication and encryption.

References

Eberspächer, J. (2017). MQTT – the machine-to-machine connectivity protocol. In Internet of Things with ESP8266 (pp. 17–34). Birmingham, UK: Packt Publishing Ltd.

Eclipse Foundation. (2021). MQTT – Overview. Retrieved from <https://iot.eclipse.org/mqtt/>

OASIS Standard. (2013, November). Message Queuing Telemetry Transport (MQTT) v3.1.1. Retrieved from <https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>

OASIS. (2021). MQTT: A Lightweight Publish/Subscribe Protocol. Retrieved from <https://www.oasis-open.org/standards#mqtt>

W3Schools. (2021). AJAX – A Beginner's Guide. Retrieved from https://www.w3schools.com/xml/ajax_intro.asp