IJIRSS
International Journal of Innovative Research & Scientific Studies

ISSN: 2617-6548

URL: www.ijirss.com

Evaluating HTTP, MQTT over TCP and MQTT over WEBSOCKET for digital twin applications: A comparative analysis on latency, stability, and integration

Bauyrzhan Amirkhanov[1], Gulshat Amirkhanova[1], Murat Kunelbayev[1,2], Saltanat Adilzhanova[1], Miras Tokhtassyn[1*]

[1]*Al-Farabi Kazakh National University, Kazakhstan.*
[2]*Institute of Information and Computational Technologies, Kazakhstan.*

Corresponding author: Miras Tokhtassyn (*Email: mirastoktasyn30@gmail.com*)

Abstract

This study aims to identify the most suitable IoT communication protocol for a digital twin architecture, focusing on meeting the demands of real-time data transmission with stability and security. The research methodology involves a comparative performance analysis of MQTT over TCP, MQTT over WebSocket, and HTTP protocols, evaluating their latency, connectivity stability, and IoT application feasibility using an ESP32 microcontroller and DHT22 sensor setup. The experimental results demonstrate that MQTT over TCP achieves the lowest average latency at 290.5 ms, while MQTT over WebSocket exhibits more stable latency profiles with a 193.2 ms standard deviation; HTTP, despite its broader compatibility, showed higher average latency of 342.6 ms with a 307.9 ms standard deviation. These findings provide valuable insights for digital twin implementations, enabling developers to make informed protocol selections based on specific requirements, whether prioritizing real-time performance through MQTT protocols or emphasizing system compatibility and simplicity through HTTP.

Keywords: Digital twin, ESP32, HTTP, MQTT, WebSocket.

DOI: 10.53894/ijirss.v8i1.4414
Funding: This research is supported by Committee of Science of the Ministry of Science and Higher Education of the Republic of Kazakhstan (Grant number: BR24992975).
History: Received: 29 November 2024/**Revised:** 6 January 2025/**Accepted:** 15 January 2025/**Published:** 31 January 2025

Publisher: Innovative Research Publishing

1. Introduction

The Internet of Things has catalyzed the creation of interconnected devices capable of sharing real-time information with users and other devices [1]. This has led to tremendous changes in several divisions, such as industry, healthcare, and intelligent infrastructure. One noticeable development in this respect is the concept of the digital twin — virtual representations of tangible entities — that enable real-time monitoring and modeling of complex systems [2].

Hypothesis: Message Queuing Telemetry Transport (MQTT) over Transmission Control Protocol (TCP) and WebSocket is less latency and have better connection than Hypertext Transfer Protocol (HTTP) in Internet of things (IoT) systems for data transfer, so it's more suitable for Digital Twin (DT) [3-6]. However, WebSocket has a bidirectional structure that fits web browsers naturally and is suitable for data visualization and web interaction tasks, which gives it a trade-off between low latency and web compatibility. HTTP, on the other hand, is good for time-to-time data transfer as well as for cloud computing, but it is worse in terms of latency and stability.

This work compares the performance of three of the most used protocols for digital twin applications: HTTP, MQTT over TCP and MQTT over WebSocket, with special attention to the evaluation of their suitability in terms of latency, connection stability, and interoperability with other systems. In this work, an ESP32-WROOM microcontroller and a Digital Humidity and Temperature 22 (DHT22) sensor were used for environmental data collection, which was transmitted via MQTT over TCP and MQTT over WebSocket to a computer configured as a broker-server using the Mosquitto MQTT broker, while a Python script was employed to measure and log latency for HTTP transmission [7, 8]. In this respect, data transmission was realized by MQTT over TCP and by MQTT over WebSocket; this allowed a detailed comparative analysis of their respective benefits and disadvantages for digital twin applications.

Figure 1.
DHT22 temperature and humidity sensor.

Figure 2.
ESP32-WROOM microcontroller.

Figure 1 and 2 represents the DHT22 temperature and humidity sensor used in the experimental setup to gather environmental data and the ESP32-WROOM microcontroller, which serves as the data publisher and is connected to the DHT22 sensor.

2. Literature Review

1. Oliveira et al. present the performance comparison of MQTT and WebSocket protocols in IoT applications using an ESP8266 microcontroller and Node.js server setup. The comparative performance evaluation of the protocols has focused on round-trip time, memory consumption, and payload size in view of the demand for low-latency protocols for IoT. Latency was measured by determining the time difference for packet exchanges between the server and ESP8266. From this, it follows that WebSocket is more suitable for applications requiring low latency with RTT of less than 1 ms. Both protocols were found viable in the context of IoT. At the same time, the relatively lower latency of WebSocket made it more suitable for real-time applications despite the benefits of MQTT in lightweight operation and community support [9].
2. In addition, Mijovic, et al. [10] reviewed application layer protocols for IoT based on STM32F411RE - Società Generale Semiconduttori (SGS) and Thomson Microelectronics 32F411RE microcontroller and ESP8266 as wireless module. They compared CoAP, WebSocket, and MQTT protocols by using an RTT algorithm and developed a coefficient which they called "efficiency." Despite using similar approach in calculating RTT as done in this study, the results are different due to differences in hardware architecture and the network layout. Further, the libraries that implement these protocols were not analyzed, which sets this work apart [10].
3. Silva et al., compared HTTP, MQTT, and WebSocket for data transmission in electric vehicle (EV) charging stations. Their work focused on latency, jitter, and throughput and for analysis, they used Wireshark. The researchers discovered that HTTP offered the greatest throughput, MQTT of the lowest latency and jitter for real-time and periodic data, and WebSocket of lower throughput but higher adaptability for real-time communications. The conclusions drawn point to the essential compromises in the choice of protocols in the context of particular uses in EV ecosystems [11].
4. In different real-time IoT scenarios, Imtiaz et al., for example, examined MQTT and WebSocket protocols. One work achieved an air flow control on AWS IoT Core via MQTT over WebSocket with low latency and high reliability. Some works assessed MQTT brokers in cloud environments, demonstrating the growth of performance as the system's complexity rose. Realistic traffic control systems also confirmed these protocols for high-frequency low latency IoT applications. This research builds upon these findings by comparing MQTT and WebSocket for real-world applications using ESP32 microcontrollers and DHT22 sensors [12].
5. MQTT as pointed out by Lakshminarayana et al. is known for its lightweight, high performance and suitability to use in scenarios where resources are limited such as in healthcare and smart city. This protocol is more efficient than HTTP in low latency and reliable communication applications. However, it does not have built-in security measures and it is prone to DoS (Denial of Service) attacks, unauthorized access, and MITM (Man-in-the-Middle). It is important to manage these risks through the right configurations, encryption and improving detection mechanisms for large scale safe IoT implementations [13].
6. According to the survey made by Mishra et al., MQTT is a low overhead, energy efficient protocol that can be used in M2M and IoT systems. Its publish/subscribe architecture allows it to run reliably over low bandwidth networks, and key industries such as healthcare, smart city, and logistics. However, plaintext data transfer with an optional encryption with TLS is a weakness that must be worked on for secure IoT settings [14].
7. Gao et al. demonstrated the use of WebSocket in a digital twin-based office management system. The system integrated IoT and deep learning to optimize resource usage and personnel detection. Real-time data acquisition was performed using ESP8266 modules, while WebSocket facilitated bi-directional communication between physical and virtual spaces. This approach enhanced energy efficiency, cost-effectiveness, and real-time control in office environments, highlighting WebSocket's utility in synchronizing digital twins [15].
8. Khan et al. proposed the Wi-Fi CSI-based digital twin model of the patient respiration using machine learning. Applying the signal processing methods of PCA (Principal Component Analysis) and EMD (Empirical Mode Decomposition), the system has high classification accuracy and provides a possibility for non-contact healthcare monitoring. This paper highlights how digital twins contribute to improving Healthcare 4.0, handling privacy issues, and controlling data noise in the IoT environment [16].
9. Last but not least, the "makeTwin" platform, described in the work by Tao et al., suggests a concept for constructing digital twins based on a set of modular components. An ability to process data in real-time, use simulation, and predict maintenance needs make it versatile for domains like manufacturing or smart cities. The layers of both physical and virtual structure are designed to be modular which enhances flexibility and scalability. Thus, the future work should focus on the issues related to the standardization and security to extend the use of the concept of digital twins in different fields [17].

3. Bibliographic Review

3.1. MQTT: Multipurpose Internet of Things Protocol

Message Queue Telemetry Transport is a lightweight messaging protocol that, from its design point of view, best suits resource-constrained devices and networks characterized by low bandwidth. Hence, because of this particular reason, MQTT is best suited for IoT applications. MQTT, relying on a publish-subscribe model, allows clients to share data in an effective manner with the help of a broker who forwards the messages to other different devices. In fact, one of the key strong points of MQTT is its flexibility because it can be deployed on top of various transport protocols, such as TCP, WebSocket, and even Secure Sockets Layer-SSL, or Transport Layer Security-TLS if one wants to securely deliver data [18, 19]. This performance capability has made MQTT a preferred option in IoT ecosystems, especially within environments that demand very low latency and low power consumption. It also supports QoS levels that allow users to define how reliably messages need to be delivered, which is crucial in many mission-critical applications [20].

Coupled with its flexibility, the lightweight design of MQTT ensures that the consumption of resources is at a minimum-a fact quite useful for microcontroller-based IoT devices. It has been illustrated that the efficiency of MQTT may outperform conventional protocols like HTTP for applications where there is a requirement for frequent transmissions of small extents of data. In the development of MQTT, its implementations were extended for various use cases such as real-time analytics, remote monitoring, and digital twin applications where low latency, reliable connectivity, and low power consumption are desired [21].

3.2. WebSocket: Real Time Communication Protocol

The development of WebSockets has been designed with the aim of providing a full-duplex communication channel over a single long-lived connection between client and server, optimized for real-time data transfer applications. Contrary to HTTP, which relies on successive requests by the client in order to obtain refreshes of an information set, WebSocket creates a long-lived connection that enables the server side to push data to the client whenever changes occur. The reasons are its low-latency and event-driven design that make it ideal for use cases such as online gaming, financial trading platforms, and IoT systems requiring fast exchanges of data. WebSocket enables web-based applications to talk directly with MQTT brokers with its implementation in MQTT, subscribing to and publishing messages within the IoT ecosystem by the clients using the web [22].

This extensibility of compatibility extends the MQTT utility into the digital twin interfaces on web platforms, improving the integration of IoT data into such platforms. However, WebSocket performance may change with variations in network conditions, especially from latency and connection stability perspectives. It has also been noted by researchers that, although WebSocket is efficient on the whole, it has greater overhead than TCP when the application requires low latency, due to the extra packet encapsulation layer introduced by WebSocket [23].

3.3. TCP: Reliable Data Transmission Backbone

Transmission Control Protocol (TCP) is one of the two core protocols comprising the Internet Protocol (IP) suite, well known for its reliable, connection-oriented data transmission [5].

Unlike UDP, which sacrifices reliability for speed, TCP ensures that the packets come out in the order they were sent. The features of error-checking and acknowledgement in TCP ensure reliable data transfer, retransmitting packets that are lost. This makes TCP quite good to go with applications where integrity of data is more vital than a little higher latency. TCP usually acts as the default transport protocol for MQTT in many situations. It provides stable connectivity, which is highly desirable in IoT environments when consistency in the data becomes key. In digital twin applications, MQTT over TCP ensures that all data synchronization from the physical systems is done completely to their virtual twin models, thereby supporting real-time analysis with almost no data loss [24].

Where the TCP is much more prone to latency compared with WebSocket in the case of high-frequency communications, which may also affect its applications that need real-time processing. Therefore, for IoT deployments with demands for quick response times with acceptable data loss, alternative protocols or optimizations such as TCP acceleration may be deployed [25].

3.4. HTTP: The Foundation of Web Communication

HTTP is a stateless, request-response protocol widely used for data exchange on the web. Unlike WebSocket, which maintains a persistent connection, HTTP establishes a new connection for each request, resulting in higher latency and overhead. This makes HTTP less suitable for real-time applications but ideal for scenarios requiring simplicity and compatibility with web-based systems [26].

In IoT contexts, HTTP is commonly used for periodic data uploads or integration with REST APIs - Application Programming Interfaces [27]. While it offers encryption and authentication when used with HTTPS, its latency and connection overhead limit its performance in digital twin systems requiring continuous synchronization [28].

4. Materials and Methods

In this regard, at al-Farabi Kazakh National University, in our laboratory Big Data and AI, we proposed one digital twin architecture for real-time monitoring and integration of data in IoT applications. Optimal IoT communication protocols that could further support the needs of the architecture and bridge the gaps in low latency, stable connectivity, and efficient data handling also form part of this very research in full flow. Basically, a perfect digital twin would represent real conditions reliably, with only minor latency, for better control and analysis of complex systems.

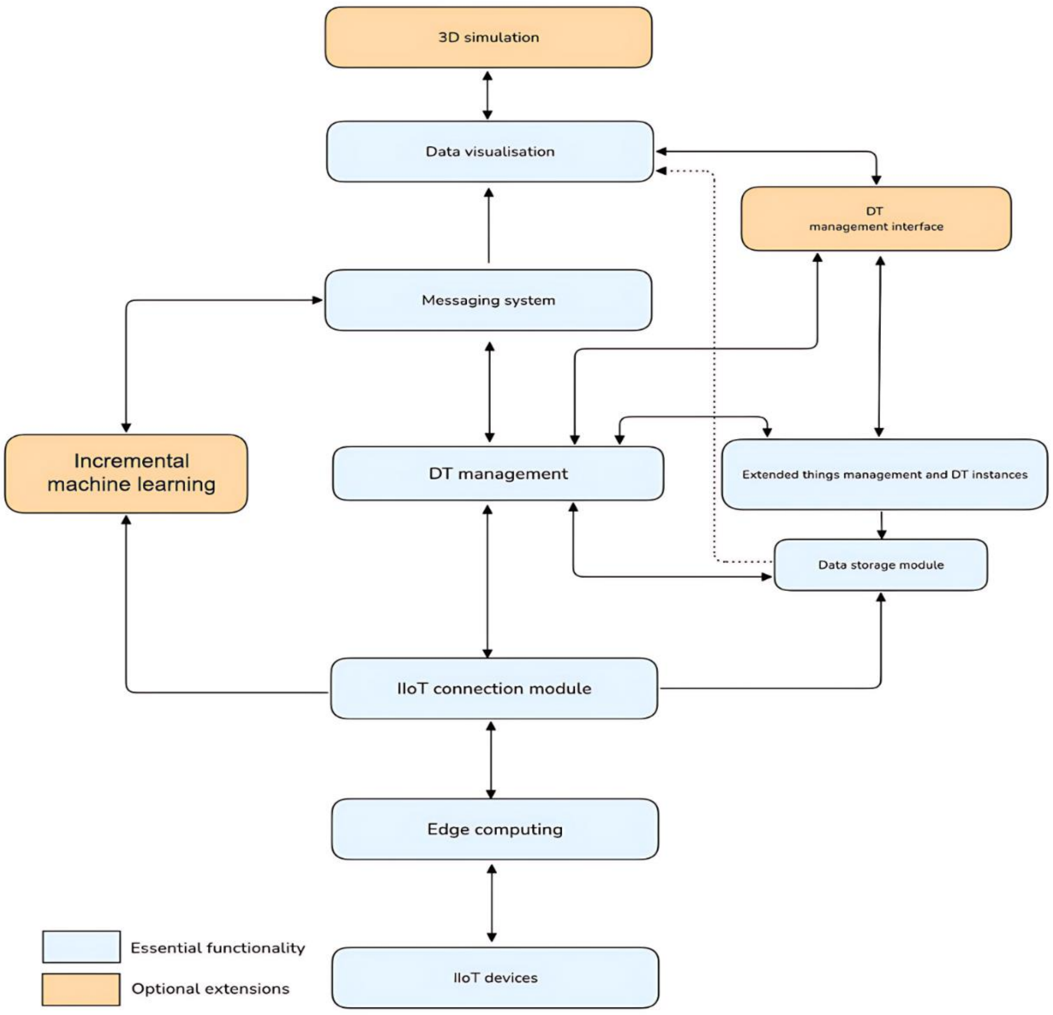


Figure 3.
The developed digital twin architecture.

Figure 3 represents our architecture for DT. This DT works by acquiring data from Industrial Internet of Things (IIoT) devices with various kinds of sensors, including temperature, pressure, and humidity, within an industrial environment. This kind of data would be routed over a connected module to the edge computing layer for preliminary processing. The output of processing goes to the IIoT connection module, which further routes the data across three main modules: the incremental machine learning module for real-time data analysis, the DT management module for coordination and managing operations of digital twins, and the data storage module for long-term and time-series data preservation.

The data are treated by the visualization module for user interaction. Real-time and historical information of the digital twin could be monitored and controlled by users through a 3D simulation interface: users can interactively and comprehensively see the performance and state of the digital twin.

The scientific novelty of this paper lies in a focused comparative analysis of MQTT over TCP and MQTT over WebSocket specifically for digital twin application. In contrast to general studies of MQTT protocols, this study examines how these two protocol variants perform on critical metrics for digital twins such as latency stability, connection reliability, and ease of integration into web applications. Using a controlled experimental setup with ESP32 and DHT22 sensors to evaluate these protocols in real-time data transmission, the study provides insights that are directly relevant to the practical deployment of digital twins in IoT. This approach helps to fill the knowledge gap to understand which protocol is best suited for digital twins, especially as IoT applications increasingly require stable, low-latency, and web-enabled solutions.

It consisted of an ESP32-WROOM microcontroller configured as a data publisher interfaced with a DHT22 temperature and humidity sensor. The broker and subscriber were composed of a computer with a Ryzen 5 3550H processor, 16GB RAM, and Realtek 8821CE LAN 802.11ac installed with Ubuntu 22.04. Both the ESP32 and the computer were connected to a Xiaomi Mi Wi-Fi Router 4A, which illustrated in Figure 4 for stable network communication.

Figure 4.
Xiaomi Mi Wi-Fi Router 4A.

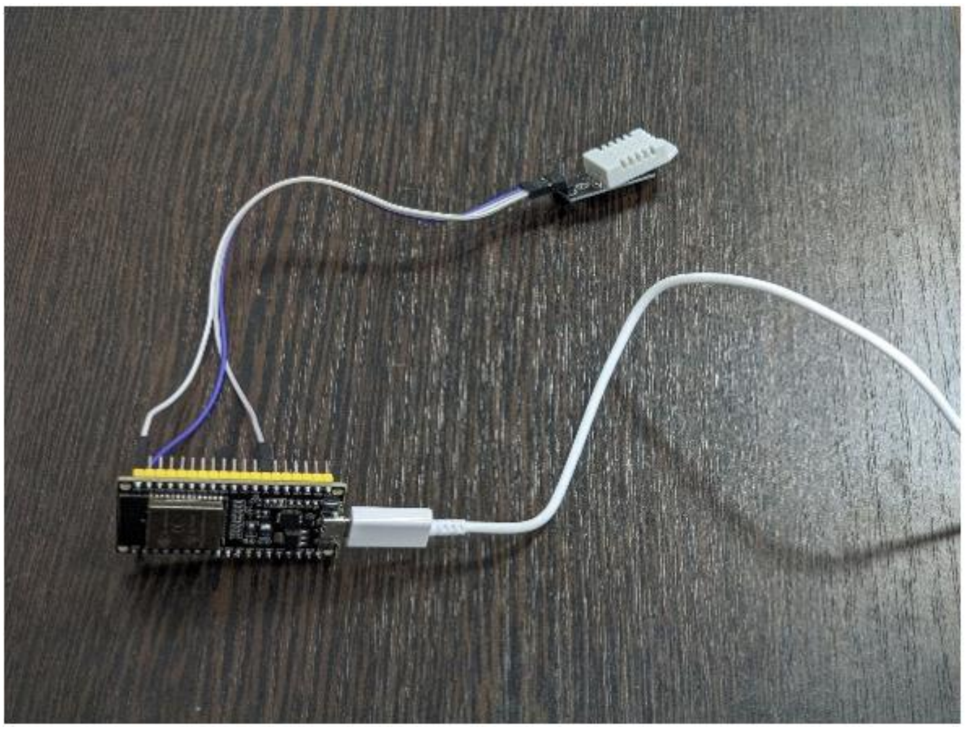


Figure 5.
The configuration of an ESP32 connected to a DHT22 sensor and a power source.

Figure 5 depicts the configuration of the ESP32 connected to the DHT22 sensor and power source, essential for setting up the MQTT data transmission.

The ESP32 was programmed with the ESP-IDF for software and libraries. Further functionality was added by utilizing the esp-idf-lib library, to get up and running with the DHT22 sensor. The computer was configured with the Mosquitto MQTT broker, that allows MQTT over TCP at port 1883 and also MQTT over WebSocket at port 9001. During the capture of the latency data, the utilization of the mosquitto_sub command piping into the ">>" writing tool from the moreutils package allowed data to be saved directly into a text file.

This Figure 6 illustrates the configuration used to measure latency in an IoT setup. Data from a DHT22 sensor is transmitted to an ESP32 microcontroller, which sends the information via Wi-Fi to a computer running the Mosquitto MQTT broker. The latency data is recorded by the computer and saved as a text file using the moreutils tool, specifically the ">>" command, for further analysis.

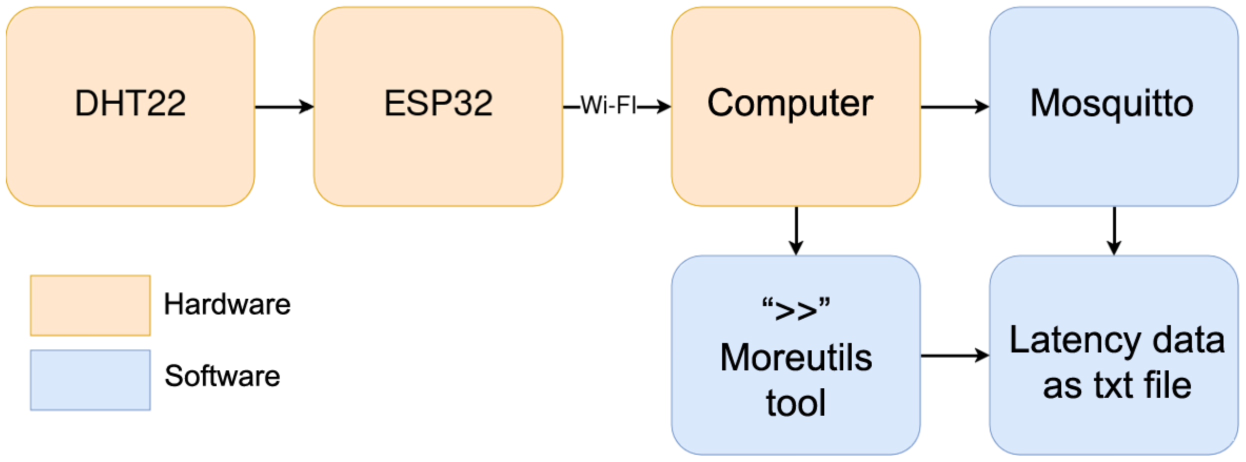


Figure 6.
Experimental setup for latency measurement using MQTT protocol.

In addition to the MQTT setup, HTTP was used as a comparative protocol for data transmission. The ESP32 was configured to send sensor data as HTTP POST requests to a simple HTTP server hosted on the same computer. The server processed the incoming data and recorded the timestamps of received requests. A Python script was utilized to measure and log the round-trip latency for HTTP requests, saving the results into a text file for analysis. This setup enabled direct comparisons between MQTT and HTTP in terms of latency, stability, and overall performance.

Figure 7 illustrates the configuration used to measure latency in an IoT setup for HTTP. A Python script captures and logs the latency of HTTP POST requests, enabling the evaluation of HTTP alongside MQTT protocols.

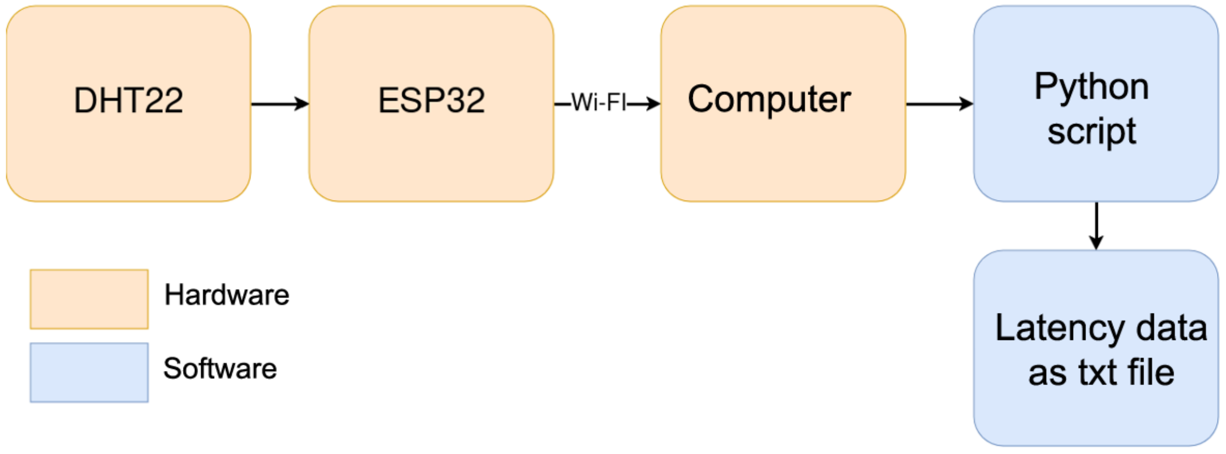


Figure 7.
Experimental setup for latency measurement using HTTP protocol.

This setup has been implemented on the ESP32, which transmits data from the DHT22 using HTTP, MQTT over TCP, and MQTT over WebSocket. Example code from the ESP-IDF framework was utilized and altered for both MQTT protocols. Initial configurations were done through the ESP-IDF menuconfig to ensure common settings across the two. A slightly different implementation was adopted for HTTP - it fit perfectly with the use case of sending data over HTTP POST requests. Data transmission times were measured in microseconds; therefore, it allowed a straightforward and meaningful comparison among all three protocols.

For MQTT tests, the broker and subscriber are the same: getting data from the ESP32 via Mosquitto. Both MQTT over TCP and MQTT over WebSocket are going to be exactly the same for a proper comparison. Their performance for each of these protocols was measured by latency, captured in real time and saved into a text file, timestamped in microseconds.

The algorithm of Figure 8 summarizes how the measurement of the latency will be done in transmitting data from an ESP32 microcontroller to a server. First, all the components which shall be used in the experiment are initialized and the timestamp of the instant is recorded when data was published. In MQTT protocols, the published data-for instance, temperature or humidity from a DHT22 sensor-is sent to the broker while in HTTP it would have been a POST request to a server.

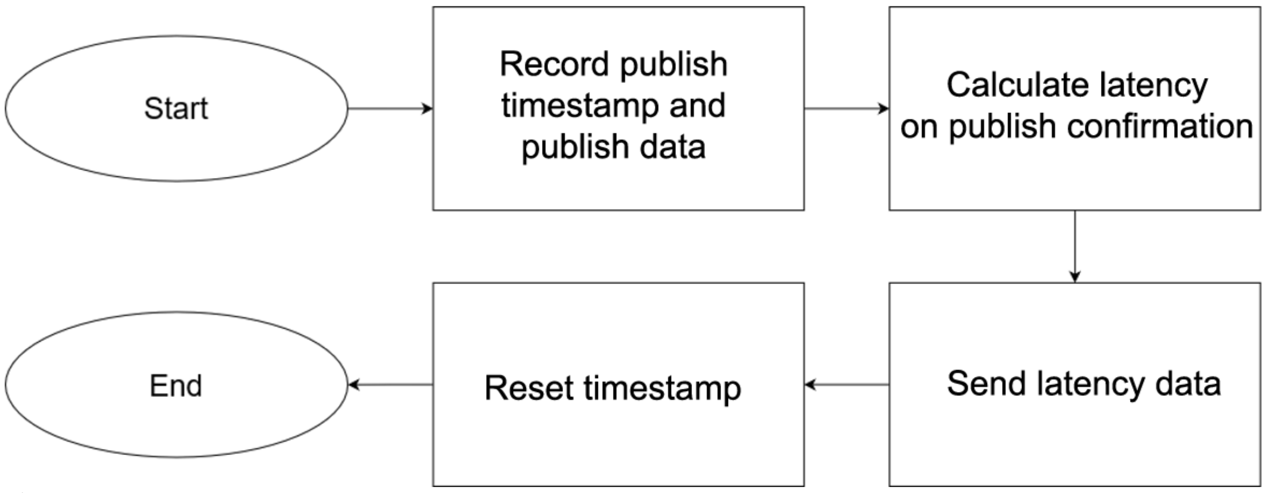

Start
Record publish timestamp and publish data
Calculate latency on publish confirmation
Send latency data
Reset timestamp
End

Figure 8.
Algorithm for measuring latency.

Figure 9 provides code that initiates an MQTT application and configures it through a secure, private connection for data publishing. The "CONFIG_BROKER_URI" variable is configured by ESP-IDF Espressif IoT Development Framework configuration panel. The values of "username" and "authentication.password" authentication data is taken from mosquitto, if there is a one.

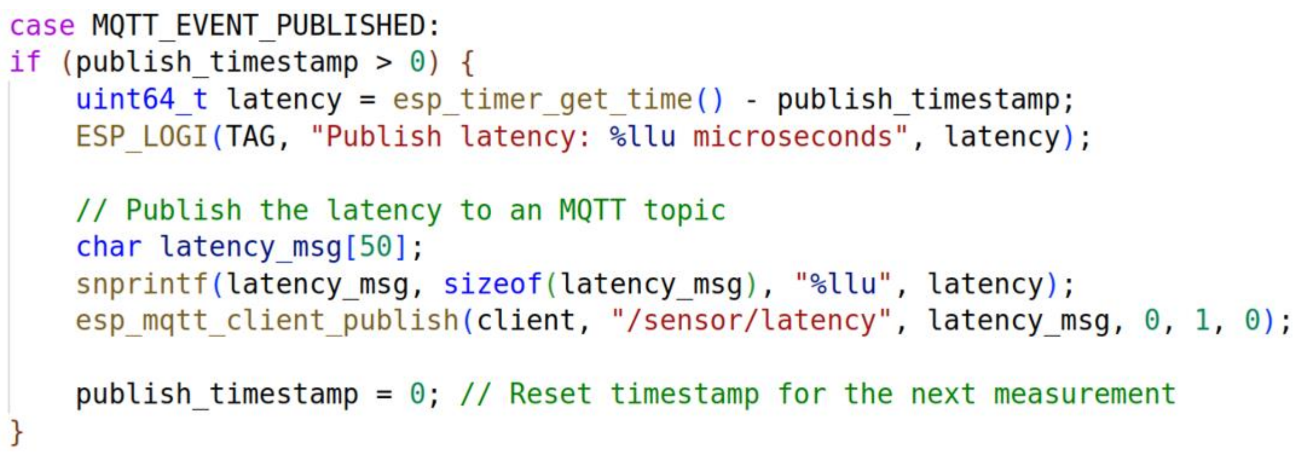
```
static void mqtt_app_start(void)
{
    const esp_mqtt_client_config_t mqtt_cfg = {
        .broker.address.uri = CONFIG_BROKER_URI,
        .credentials.username = "micola",
        .credentials.authentication.password = "andromeda",
    };

    client = esp_mqtt_client_init(&mqtt_cfg);
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
    esp_mqtt_client_start(client);
}
```

Figure 9.
This code initiates an MQTT application and sets it up using a secure, private connection.

The script will read the temperature and humidity data through an ESP32 microcontroller fitted with a DHT22 sensor every 2 seconds and send the data across via MQTT. It initiates after setting up Wi-Fi and after it has connected to the broker, reading data from the sensor, converting the readings, and Figure 10 shows the code of publishing them on specific MQTT topics for temperature (/home/temperature) and humidity (/home/humidity). Each publish action returns a timestamp, allowing the algorithm to quantify latency by computing the difference between data send and broker acknowledgement. This calculated latency is, in turn, published to a dedicated topic (/sensor/latency) for real-time performance monitoring, the code of which represented in Figure 11. Both MQTT over TCP and MQTT over WebSocket have been tested in order to compare various protocols under the same conditions.

```
// Publishing temperature
snprintf(msg, sizeof(msg), "%.1f", temperature_f);
publish_timestamp = esp_timer_get_time();  // Record the publish timestamp
esp_mqtt_client_publish(client, "/home/temperature", msg, 0, 1, 0);

// Publishing humidity
snprintf(msg, sizeof(msg), "%.1f", humidity_f);
publish_timestamp = esp_timer_get_time();  // Record the publish timestamp
esp_mqtt_client_publish(client, "/home/humidity", msg, 0, 1, 0);
```

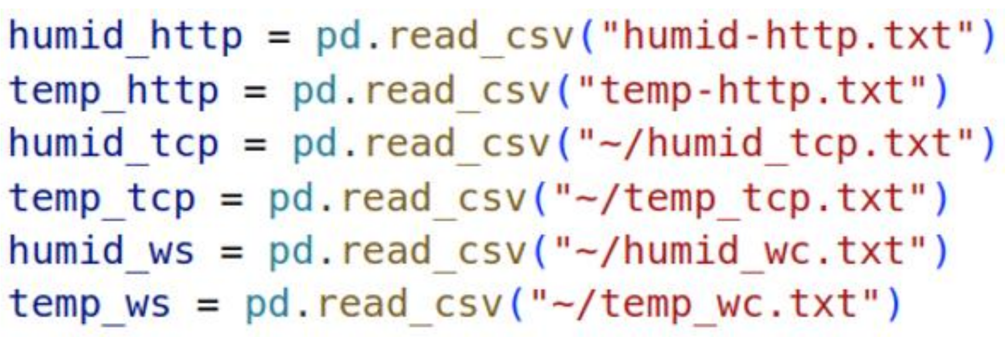
Figure 10.
This code manages publishing temperature and humidity data on topic "/home/temperature" and "/home/humidity".



```c
case MQTT_EVENT_PUBLISHED:
if (publish_timestamp > 0) {
    uint64_t latency = esp_timer_get_time() - publish_timestamp;
    ESP_LOGI(TAG, "Publish latency: %llu microseconds", latency);

    // Publish the latency to an MQTT topic
    char latency_msg[50];
    snprintf(latency_msg, sizeof(latency_msg), "%llu", latency);
    esp_mqtt_client_publish(client, "/sensor/latency", latency_msg, 0, 1, 0);

    publish_timestamp = 0; // Reset timestamp for the next measurement
}
```

Figure 11.
This code publishes latency data to the topic "/sensor/latency" upon receiving confirmation of successful publishing of DHT22 data.

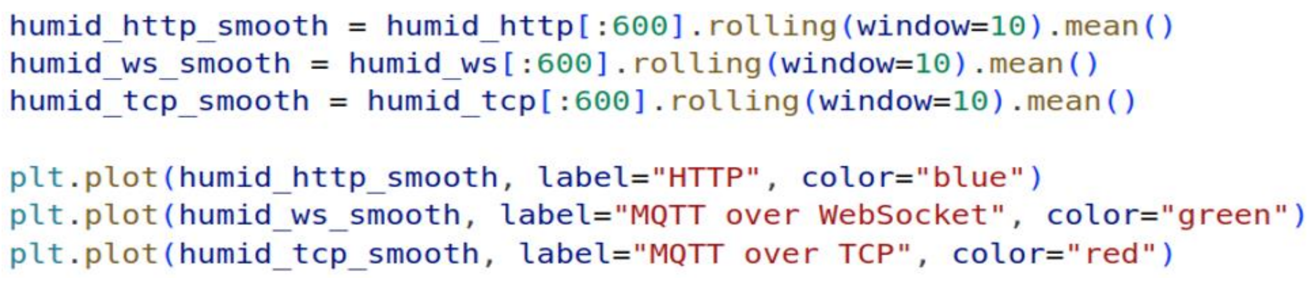
The statistic values of latency data were obtained using Python with the Pandas and MatPlotLib libraries. When plotting humidity and temperature data, the rolling method was used with window value 10 because of spikes that appeared due to a small variation of values, as the room had a stable climate.



```python
humid_http = pd.read_csv("humid-http.txt")
temp_http = pd.read_csv("temp-http.txt")
humid_tcp = pd.read_csv("~/humid_tcp.txt")
temp_tcp = pd.read_csv("~/temp_tcp.txt")
humid_ws = pd.read_csv("~/humid_wc.txt")
temp_ws = pd.read_csv("~/temp_wc.txt")
```

Figure 12.
This code imports data from txt files.

Figure 12 represents the initial data loading process using the Pandas library in Python. Here, data files for humidity and temperature values recorded over TCP and WebSocket protocols are loaded into separate variables (humid_tcp, humid_ws, temp_tcp, temp_ws) for further processing and analysis.

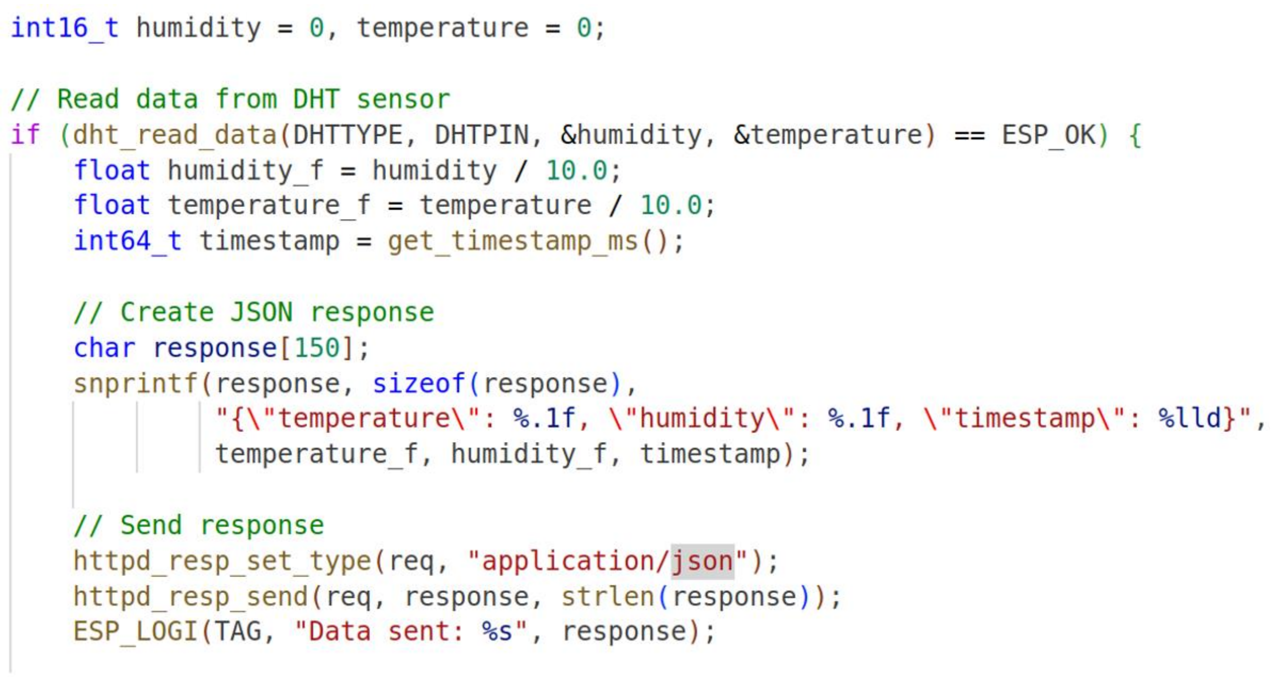


```python
humid_http_smooth = humid_http[:600].rolling(window=10).mean()
humid_ws_smooth = humid_ws[:600].rolling(window=10).mean()
humid_tcp_smooth = humid_tcp[:600].rolling(window=10).mean()

plt.plot(humid_http_smooth, label="HTTP", color="blue")
plt.plot(humid_ws_smooth, label="MQTT over WebSocket", color="green")
plt.plot(humid_tcp_smooth, label="MQTT over TCP", color="red")
```

Figure 13.
This code plots humidity data obtained through MQTT over WebSocket.

Figure 13 illustrates the application of a rolling mean to smooth out the temperature or humidity data. A window size of 10 is used to average adjacent values, reducing spikes caused by minor variations in a stable environment. This smoothing technique helps in visualizing general trends without the noise from frequent small fluctuations, making the data easier to analyze. The smoothed data is then plotted with adjusted figure size and font for clarity.

As shown in the Figure 14 HTTP application of ESP32 is programmed to fetch environment data from the DHT22 sensor and further send this data as a response to the client in JSON - JavaScript Object Notation format. It first starts with readings of temperature and humidity through the "dht_read_data" function of the sensor. Then it converts the data into a JSON response in such a way that temperature, humidity, and timestamp get packed together in a format understandable by man and machine.

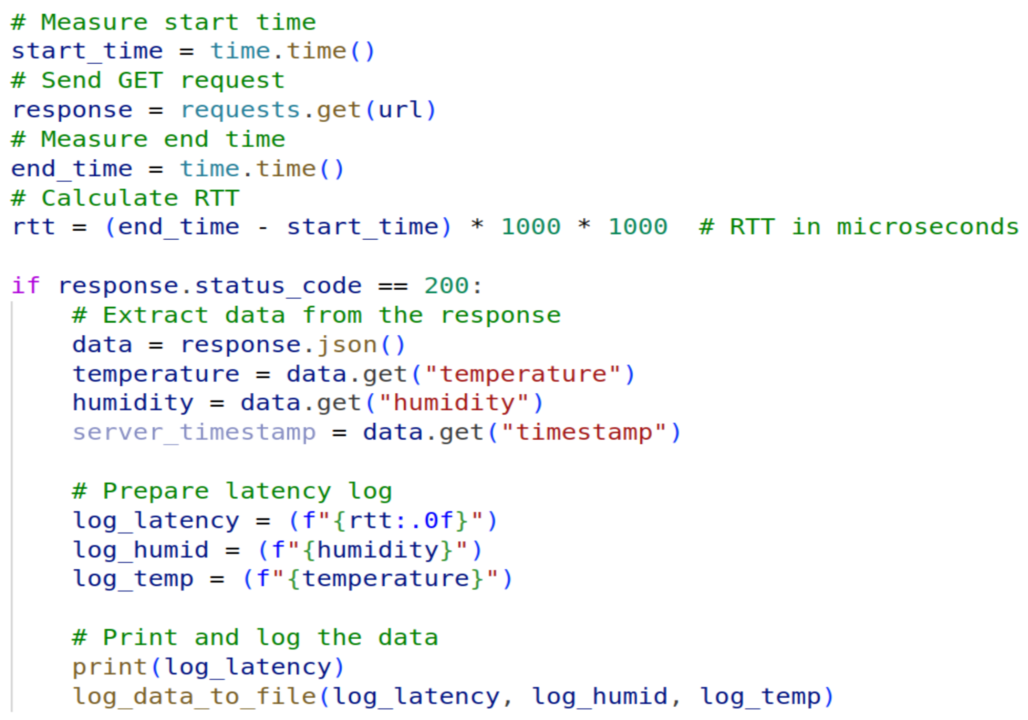


```c
int16_t humidity = 0, temperature = 0;

// Read data from DHT sensor
if (dht_read_data(DHTTYPE, DHTPIN, &humidity, &temperature) == ESP_OK) {
    float humidity_f = humidity / 10.0;
    float temperature_f = temperature / 10.0;
    int64_t timestamp = get_timestamp_ms();

    // Create JSON response
    char response[150];
    snprintf(response, sizeof(response),
             "{\"temperature\": %.1f, \"humidity\": %.1f, \"timestamp\": %lld}",
             temperature_f, humidity_f, timestamp);

    // Send response
    httpd_resp_set_type(req, "application/json");
    httpd_resp_send(req, response, strlen(response));
    ESP_LOGI(TAG, "Data sent: %s", response);
```

Figure 14.
This code sends data through HTTP.

Finally, the HTTP response is returned to the client with a predefined content type, "application/json", defining that the data is in JSON format. That way, the ESP32 can behave like an HTTP server and provide sensor readings upon request.

The Python script, which demonstrated in Figure 15 is designed to measure and log the round-trip time (RTT) of HTTP requests sent to the ESP32 server, which provides environmental data from a DHT22 sensor. It starts by recording the current time, "start_time", just before making an HTTP GET request to the server using the "requests.get()" function. It records the time again once the response comes in, as would be expected, and calculates the round-trip time in microseconds by subtracting the start time from the end time. The script logs the RTT, temperature, and humidity data to both the console and to a file using a custom function responsible for logging data into a file "log_data_to_file". In this way, all the latency measurements together with sensor readings would be saved for further analysis, thus allowing detailed comparisons of HTTP performance against other protocols such as MQTT. This provides a simple yet effective method of evaluating efficiency and reliability in HTTP for IoT systems.



```python
# Measure start time
start_time = time.time()
# Send GET request
response = requests.get(url)
# Measure end time
end_time = time.time()
# Calculate RTT
rtt = (end_time - start_time) * 1000 * 1000  # RTT in microseconds

if response.status_code == 200:
    # Extract data from the response
    data = response.json()
    temperature = data.get("temperature")
    humidity = data.get("humidity")
    server_timestamp = data.get("timestamp")

    # Prepare latency log
    log_latency = (f"{rtt:.0f}")
    log_humid = (f"{humidity}")
    log_temp = (f"{temperature}")

    # Print and log the data
    print(log_latency)
    log_data_to_file(log_latency, log_humid, log_temp)
```

Figure 15.
This code reads data from ESP32 server and logs as txt file.

Testing of the protocols was based on three important metrics: latency, stability of connection, and integrative capabilities. Averaged and peak latency for each protocol was calculated based on recorded data. Connection stability was assessed based on drop rates and duration of reconnections observed during tests. The interoperability of the protocols was further investigated by considering MQTT standards in the case of MQTT-based protocols and assessing the flexibility in HTTP configurations for interoperability with other systems.

The performance of MQTT over TCP, MQTT over WebSocket, and HTTP was assessed through a combination of qualitative and quantitative metrics, determining the suitability of each protocol for applications targeting digital twins. The framework for the comparison was structured as follows:

4.1. Qualitative Analysis

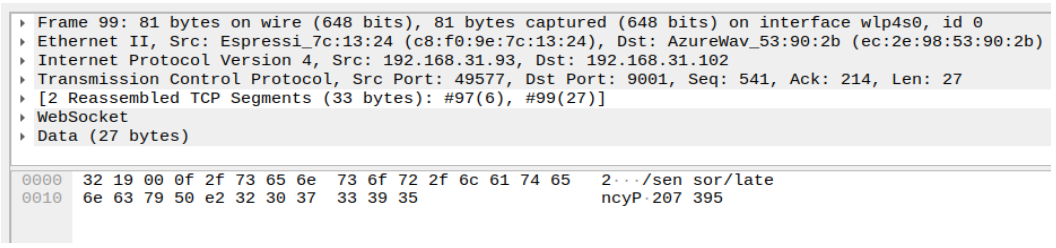
- Security.
- Complexity of implementation.
- Compatibility with Web-Based Applications.

4.2. Quantitative Tests

- Latency.
- Connection Stability.

5. Results

This section may be divided by subheadings. It should provide a concise and precise description of the experimental results, their interpretation as well as the experimental conclusions that can be drawn.



```
Frame 99: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface wlp4s0, id 0
Ethernet II, Src: Espressi_7c:13:24 (c8:f0:9e:7c:13:24), Dst: AzureWav_53:90:2b (ec:2e:98:53:90:2b)
Internet Protocol Version 4, Src: 192.168.31.93, Dst: 192.168.31.102
Transmission Control Protocol, Src Port: 49577, Dst Port: 9001, Seq: 541, Ack: 214, Len: 27
[2 Reassembled TCP Segments (33 bytes): #97(6), #99(27)]
WebSocket
Data (27 bytes)

0000  32 19 00 0f 2f 73 65 6e  73 6f 72 2f 6c 61 74 65   2···/sen sor/late
0010  6e 63 79 50 e2 32 30 37  33 39 35                  ncyP·207 395
```

Figure 16.
MQTT over WebSocket packet analysis.

5.1. Qualitative Analysis

5.1.1. MQTT over WebSocket

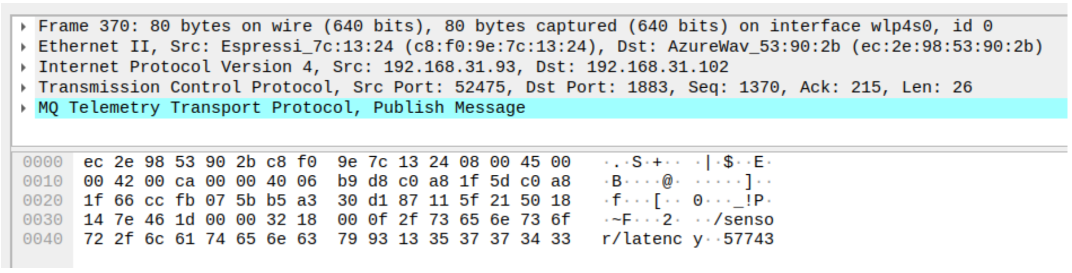
The Figure 16 shows the Wireshark capture of MQTT over WebSocket traffic. The MQTT data (e.g., /home/humidity and its payload) is encapsulated over a connection on port 9001 using the WebSocket protocol.

WebSocket framing introduces additional overhead compared to MQTT over TCP, so that packets are slightly larger and reassembled TCP segments. However, this encapsulation supports web-based integration but under some conditions may impact latency.

Websocket supports SSL/TLS encryption, which protects the data in transit using the "wss://" protocol. On the other hand, compared with normal TCP sockets, exposure of WebSocket in the web may raise additional security concerns in that it will be much more vulnerable to web-based attacks-such as XSS or CSRF-when integrated into browser applications [29, 30]. Additional complexity is added to set up SSL/TLS on top of Websocket, but this can provide adequate security for general IoT applications.

In general, MQTT over WebSocket will require more configuration than TCP due to the added layer of WebSocket protocol and its handshake processes. While WebSocket is quite well-supported within web-based systems, setting up with encryption and ensuring it will be compatible with components not based on the web adds complexity at least in an environment that does not natively support WebSocket.

WebSockets support naturally lives in web-based applications, as all web browsers natively support them. As such, it was ideal for digital twin applications that needed to interface directly with the web for easy data visualization and interaction on browser-based platforms without requiring additional middleware.



```
Frame 370: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface wlp4s0, id 0
Ethernet II, Src: Espressi_7c:13:24 (c8:f0:9e:7c:13:24), Dst: AzureWav_53:90:2b (ec:2e:98:53:90:2b)
Internet Protocol Version 4, Src: 192.168.31.93, Dst: 192.168.31.102
Transmission Control Protocol, Src Port: 52475, Dst Port: 1883, Seq: 1370, Ack: 215, Len: 26
MQ Telemetry Transport Protocol, Publish Message

0000  ec 2e 98 53 90 2b c8 f0  9e 7c 13 24 08 00 45 00   ·.·S·+·· ·|·$··E·
0010  00 42 00 ca 00 00 40 06  b9 d8 c0 a8 1f 5d c0 a8   ·B····@· ·····]··
0020  1f 66 cc fb 07 5b b5 a3  30 d1 87 11 5f 21 50 18   ·f···[·· 0···_!P·
0030  14 7e 46 1d 00 00 32 18  00 0f 2f 73 65 6e 73 6f   ·~F···2· ··/senso
0040  72 2f 6c 61 74 65 6e 63  79 93 13 35 37 37 34 33   r/latenc y··57743
```

Figure 17.
MQTT over TCP packet analysis.

5.1.2. MQTT Over TCP

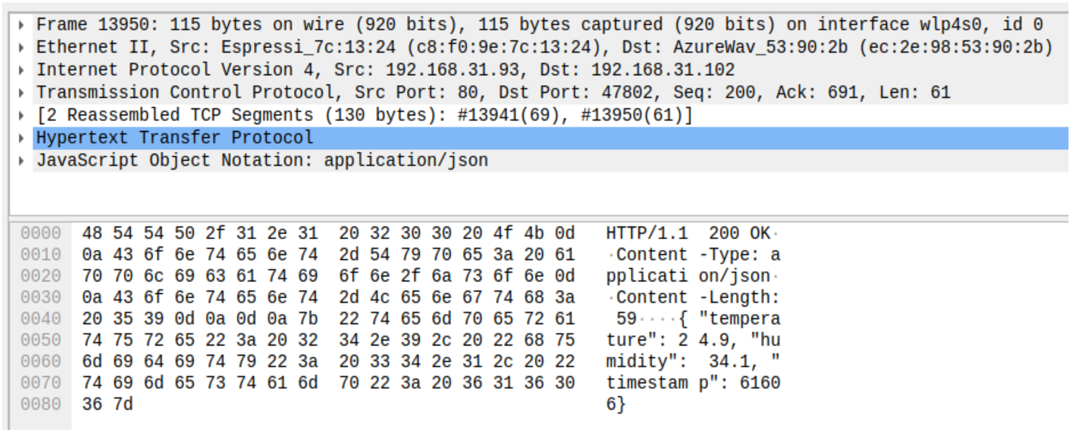
The Figure 17 is the Wireshark analysis of data sent using MQTT over TCP. The data is published on the MQTT topic "sensor/latency" and includes payload data such as latency values (e.g., "y:57743").

The analysis shows that MQTT over TCP is compact and efficient, with minimal overhead and direct transmission of the publish message. Efficient use of network resources for IoT applications is confirmed by the packet length and the use of a single TCP segment.

MQTT over TCP is usually secured with SSL/TLS, which provides strong encryption suitable for IoT environments, with the main requirement of secure and reliable data exchange. Being on the TCP level, contrary to WebSocket, it is also more secure and less exposed to web-based vulnerabilities in closed networks or private IoT deployments. Setup on SSL/TLS for TCP is straightforward and very well supported in MQTT implementations, hence even more suitable for secure digital twin applications.

In general, TCP implementation is simpler in traditional IoT networks, as it requires fewer steps in implementation and configuration compared to WebSocket. Most MQTT brokers natively support MQTT over TCP, which reduces the complexity in establishing the connections. This simplicity in use makes TCP more accessible for projects with limited resources or when rapid deployment is necessary.

While TCP is highly reliable in backend systems, most of the web browsers do not natively support it due to which middleware or other software alternatives are used for Web-based integration. This might be an impedance threshold in applications of digital twins relying on real-time interaction with the Web. However, TCP still communicates with the backend infrastructure and is thus often used together with server-based applications that do not require direct access to the Web.



```
Frame 13950: 115 bytes on wire (920 bits), 115 bytes captured (920 bits) on interface wlp4s0, id 0
Ethernet II, Src: Espressi_7c:13:24 (c8:f0:9e:7c:13:24), Dst: AzureWav_53:90:2b (ec:2e:98:53:90:2b)
Internet Protocol Version 4, Src: 192.168.31.93, Dst: 192.168.31.102
Transmission Control Protocol, Src Port: 80, Dst Port: 47802, Seq: 200, Ack: 691, Len: 61
[2 Reassembled TCP Segments (130 bytes): #13941(69), #13950(61)]
Hypertext Transfer Protocol
JavaScript Object Notation: application/json

0000  48 54 54 50 2f 31 2e 31  20 32 30 30 20 4f 4b 0d   HTTP/1.1  200 OK·
0010  0a 43 6f 6e 74 65 6e 74  2d 54 79 70 65 3a 20 61   ·Content -Type: a
0020  70 70 6c 69 63 61 74 69  6f 6e 2f 6a 73 6f 6e 0d   pplicati on/json·
0030  0a 43 6f 6e 74 65 6e 74  2d 4c 65 6e 67 74 68 3a   ·Content -Length:
0040  20 35 39 0d 0a 0d 0a 7b  22 74 65 6d 70 65 72 61    59····{ "tempera
0050  74 75 72 65 22 3a 20 32  34 2e 39 2c 20 22 68 75   ture": 2 4.9, "hu
0060  6d 69 64 69 74 79 22 3a  20 33 34 2e 31 2c 20 22   midity":  34.1, "
0070  74 69 6d 65 73 74 61 6d  70 22 3a 20 36 31 36 30   timestam p": 6160
0080  36 7d                                              6}
```

Figure 18.
HTTP packet analysis.

5.1.3. HTTP

In Figure 18, we see an HTTP POST request and response as captured in Wireshark. The server responds with an HTTP 200 status and includes temperature, humidity, and timestamp data in JSON format in the request.

The overhead of HTTP is much larger than TCP because HTTP is stateless, and each request must contain headers and reassemble multiple TCP segments. This overhead is functional for periodic data uploads, but has higher latency than MQTT.

When paired with HTTPS (SSL/TLS encryption), HTTP provides a secure and widely supported means of data transmission over the Internet, providing confidentiality and integrity of communication. HTTP is a simple protocol that is compatible with almost all web-based systems and thus a common choice for integrating IoT data with cloud platforms and web applications. But HTTP's stateless nature and the requirement to create a new connection for each request causes a significant overhead, resulting in higher latency than MQTT protocols.

HTTP is one of the key reasons why it is so popular, because all web browsers and web-based tools inherently support the protocol. As a result, HTTP is a suitable choice for digital twin applications where periodic data uploads or simple integration with web interfaces is all that is needed. Also, working with HTTP is easy, which lowers the barrier to entry for projects that don’t have a lot of IoT expertise or resources.

While HTTP is beneficial, it is not a good choice for real time communication in IoT systems. HTTP doesn’t have a persistent connection, so it can cause delays of frequent data exchange scenarios. Moreover, its performance is more sensitive to high frequency requests, where connection setup and teardown are also inefficient. Therefore, it is not necessarily the best choice for digital twin applications that require continuous, low latency synchronization of the physical and virtual systems.

In IoT and digital twin systems, HTTP is overall a simple and secure way to exchange data, especially in environments that seek simplicity, broad compatibility, or integration with existing web-based infrastructures. But its latency and connection persistence are too limited to be used in applications with strict real time requirements.

5.2. Quantitative Tests

Figures 19, 20 and 21 depict the outcomes of the latency measurements, respectively, using HTTP, MQTT over WebSocket and MQTT over TCP. By analyzing latency, trends indicate that under identical conditions, different protocols will have clear patterns of performance and, hence, latency characteristics valuable in view of the digital twin application.

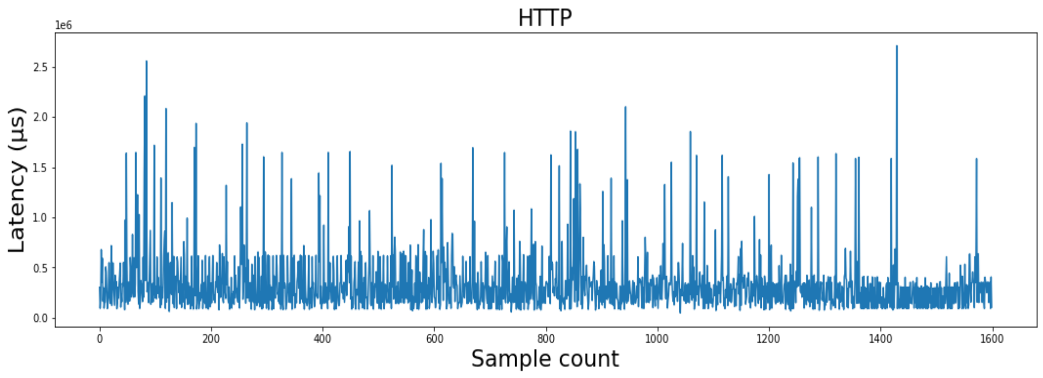


Figure 19.
Latency measurements using HTTP, showing significant variability due to repeated connection setup and teardown inherent to the protocol.

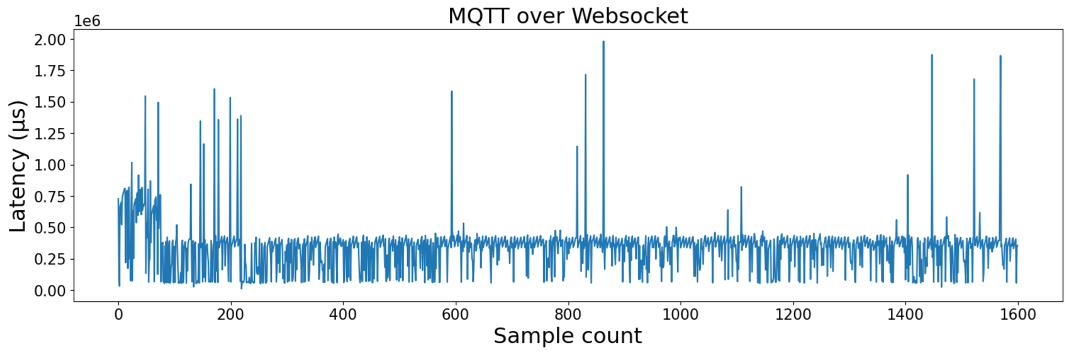


Figure 20.
Latency measurements using WebSocket, with MQTT over WebSocket showing stable latency with intermittent high spikes.

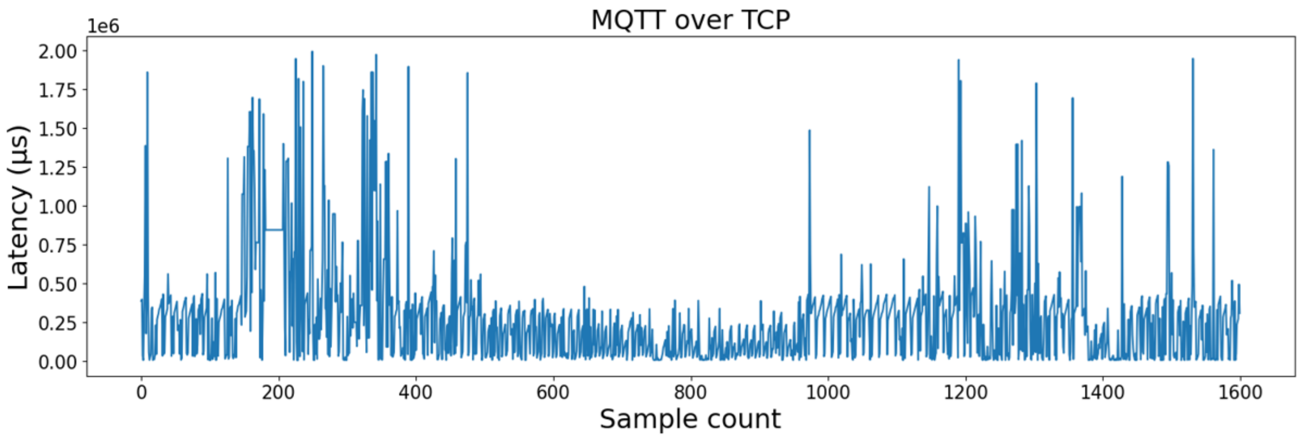


Figure 21.
Latency measurements using TCP, with MQTT over TCP displaying frequent latency spikes due to retransmission mechanisms.

The following charts give a measure of humidity and temperature over time for HTTP, MQTT over TCP and MQTT over WebSocket protocols.

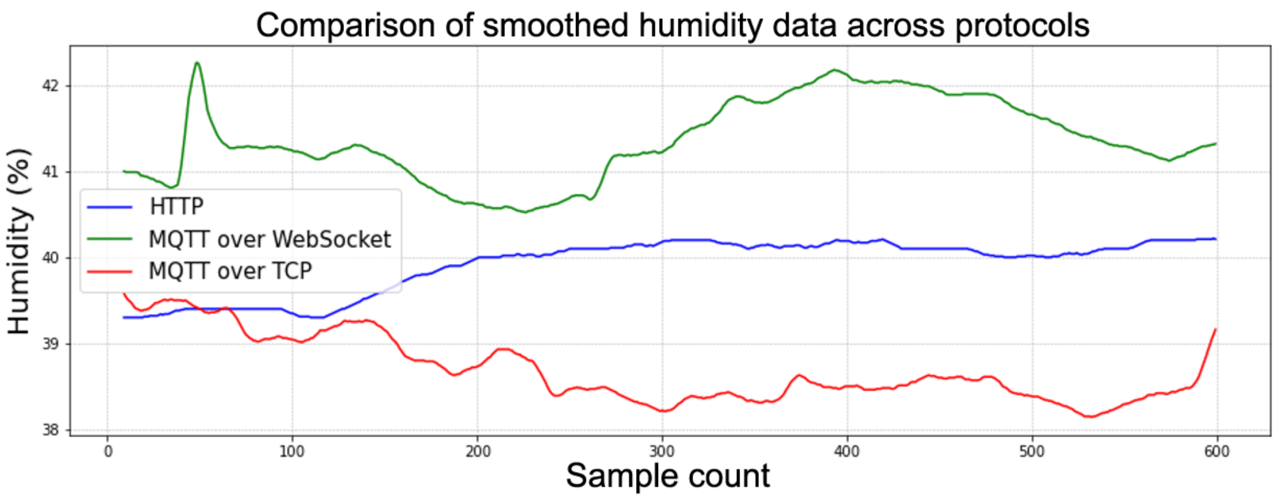


Figure 22.
Humidity levels over time.

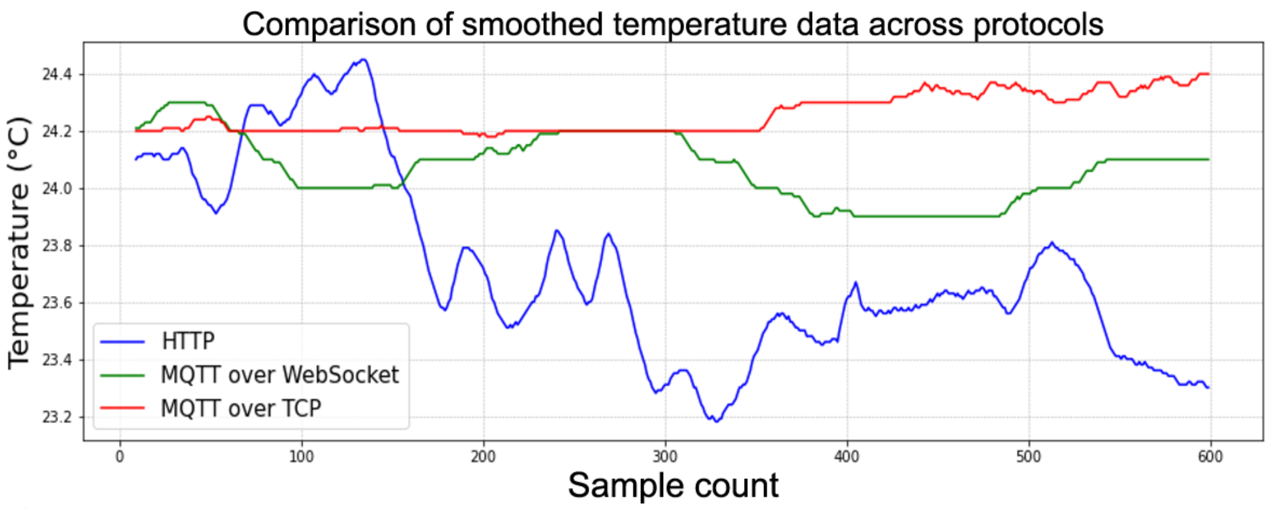


Figure 23.
Temperature levels over time.

The obtained results in Figures 22 and 23 indicate that the protocols are good in recording environmental data.

The plots shows that MQTT over WebSocket provides a more stable connection and quicker recovery after an interruption, while MQTT over TCP faces some stability problems due to the retransmission protocol.

Table 1.
Descriptive statistics values of latency data using MQTT over TCP and over WebSocket.

Protocol	**Mean**	**Median**	**Standard deviation**	**Minimum**	**Maximum**
HTTP	342.645	289.960	307.931	47.761	2707.353
WebSocket	341.973	365.273	193.238	10.159	1982.482
TCP	290.500	266.012	325.719	3.523	1992.529

Note: Values in milliseconds.

In Table 1, statistical values for latency metrics of HTTP, MQTT over WebSocket and MQTT over TCP are presented. The table provides information on the latency of protocols through mean, median, standard deviation, minimum, and maximum latency values.

Among the protocols, HTTP had the highest average latency of 342.645 ms and the highest variability of 307.931 ms of standard deviation. Its minimum latency was 47.761 ms and the maximum was 2707.353 ms, this shows that connection overhead and setup time severely affected its performance.

WebSocket with MQTT revealed less variability in latency than HTTP with a standard deviation of 193.238ms. It was able to achieve a mean latency of 341.973 ms and median latency of 365.273 ms which are quite stable. WebSocket had the least latency of 10.159 ms and the highest of 1982.482 ms with occasional high latency values.

MQTT over TCP on the other hand, recorded the least average latency of 290.500ms making it the fastest protocol on average. However, TCP had the highest variance with the standard deviation of 325.719 ms. The minimum latency observed was 3.523 ms, while the maximum latency recorded was 1992.529 ms because of retransmission mechanisms.

In total, TCP has the least average latency but has a higher standard deviation as compared to WebSocket that provides better latency consistency. HTTP is easy to understand and works with almost all devices but it is too heavy and thus not so efficient for real time use as compared to MQTT protocols.

6. Discussion

The findings of this study offer a comparative analysis of MQTT over TCP, MQTT over WebSocket, and HTTP to understand the applicability of each protocol for digital twin applications. The results are consistent with the literature and offer new interpretations specific to digital twin systems.

TCP with MQTT revealed the lowest mean latency (290.500 ms) in the experiments, which again proves its appropriateness for applications that involve the exchange of data within a short time. Nonetheless, the higher standard deviation of 325.719 ms shows that the system is more prone to latency spikes, especially due to retransmission mechanisms. These characteristics imply that, although MQTT over TCP is appropriate for backend systems and industrial applications where data synchronization is crucial, its variability may not be suitable for use cases that require high-level performance consistency. Prior works also focus on the reliability of TCP, yet this work shows that it has its flaws in real-time digital twin scenarios where latency steadiness is paramount.

WebSocket MQTT had the worst average latency of 341.973 ms but with a much lower standard deviation of 193.238 ms in comparison to TCP. This stable performance is in concordance with its design for full-duplex communication and also compatibility with the web platforms. WebSocket is highly advantageous for digital twin applications especially those that are based on browser systems as it is naturally compatible with them particularly in applications that require interactive dashboards and real time user interfaces. These results support previous studies on WebSocket's effectiveness for regular, synchronous data transfer and highlight its benefits for web-integrated digital twins.

HTTP, with the highest average latency (342.645 ms) and variability (standard deviation: 307.931ms). The slower response times of the models showed it to be less ideal for real-time digital twin use. Its stateless characteristic and the need to reconnect frequently make it less suitable for streaming data changes. However, it is quite basic, compatible with almost any web and cloud platform, useful for occasional data backup and for systems that value compatibility over speed. These findings support previous research on HTTP's shortcomings in IoT systems but broadens its application by positioning HTTP as a feasible solution for non-latency-sensitive digital twin applications that are easy to implement.

The results reveal that the choice of a protocol for digital twins should consider latency, stability, and integration constraints. For industrial monitoring and control, where fast feedback loops are required, MQTT over TCP is still the preferred one if latency peaks are tolerable. For web-based applications which focus more on the interaction and visualisation of the data in real time, MQTT over WebSocket is more appropriate because of its stable latency. HTTP is not very efficient for real-time applications but it provides a good solution for applications that require data to be updated periodically or for applications that need to be integrated with web browsers.

In future research, related authors should investigate the performance of other secure protocols, for example, HTTPS and WSS to determine their effects on latency and dependability in DT applications. Moreover, experiments with different hardware platforms, network environments, and application contexts would give further comprehension of these protocols' performance. Exploring composite protocols that integrate the features of several protocols may also improve their use in multiple-layered digital twin systems.

This research contributes to the state of the art of IoT protocols in digital twin contexts and provides a starting point for selecting and improving protocols as the use of digital twins continues to expand.

7. Conclusion

In this work, MQTT over TCP, MQTT over WebSocket, and HTTP were compared for use in digital twin with emphasis on latency, stability, and integration. MQTT over TCP had the lowest mean latency of 290.500 ms but was more variable and therefore less suitable for applications that require a higher degree of real time responsiveness. MQTT over WebSocket had a higher latency of 341.973 ms but had better stability and was suitable for web applications that require stable connection. HTTP works best for systems that require high simplicity and compatibility at the expense of low latency, which is 342.645 ms, and high variability.

The study shows that the choice of the protocol is based on the application requirements, the speed/ stability/interoperability ratio. As for future work, it is necessary to investigate the secure configurations and the combination of centralized and decentralized digital twin configurations to achieve the best results in various contexts.

References

[1] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of things (IoT): A literature review," *Journal of Computer and Communications*, vol. 3, no. 5, pp. 164-173, 2015. https://doi.org/10.4236/jcc.2015.35021

[2] F. Tao, H. Zhang, A. Liu, and A. Y. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405-2415, 2018.

[3] Y. Cho and S. D. Noh, "Design and implementation of digital twin factory synchronized in real-time using MQTT," *Machines*, vol. 12, no. 11, p. 759, 2024. https://doi.org/10.3390/machines12110759

[4] J. Postel, "Transmission control protocol," *RFC793*, 1981. https://doi.org/10.17487/rfc0793

[5] I. Fette and A. Melnikov, "The websocket protocol," 2011. https://doi.org/10.17487/rfc6455

[6] R. Fielding, *Hypertext transfer protocol--HTTP/1.1*. The Internet Society. https://doi.org/10.17487/rfc2616, 2008.

[7] G. P. Pereira and M. Z. Chaari, "Comparison of blynk IoT and ESP rainmaker on ESP32 as beginner-friendly IoT solutions," in *International Conference on Internet of Things*, 2022: Springer, pp. 123-132.

[8] J. Monteiro, J. Barata, M. Veloso, L. Veloso, and J. Nunes, "A scalable digital twin for vertical farming," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 10, pp. 13981-13996, 2023. https://doi.org/10.1007/s12652-022-04106-2

[9] G. M. Oliveira *et al.*, "Comparison between MQTT and WebSocket protocols for Iot applications using ESP8266," in *2018 Workshop on Metrology for Industry 4.0 and Io*T. https://doi.org/10.1109/metroi4.2018.8428348, 2018: IEEE, pp. 236-241.

[10] S. Mijovic, E. Shehu, and C. Buratti, "Comparing application layer protocols for the Internet of Things via experimentation," in *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI).* https://doi.org/10.1109/rtsi.2016.7740559, 2016: IEEE, pp. 1-5.

[11] D. R. Silva, G. M. Oliveira, I. Silva, P. Ferrari, and E. Sisinni, "Latency evaluation for MQTT and WebSocket Protocols: an Industry 4.0 perspective," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018: IEEE, pp. 01233-01238.

[12] N. I. Jaya and M. F. Hossain, "A prototype air flow control system for home automation using mqtt over websocket in aws iot core," in *2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, 2018: IEEE, pp. 111-1116.

[13] S. Lakshminarayana, A. Praseed, and P. S. Thilagam, "Securing the IoT application layer from an MQTT protocol perspective: Challenges and research prospects," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 4, pp. 2510–2546, 2024. https://doi.org/10.1109/comst.2024.3372630

[14] B. Mishra and A. Kertesz, "The use of MQTT in M2M and IoT systems: A survey," *IEEE Access*, vol. 8, pp. 201071-201086, 2020. https://doi.org/10.1109/access.2020.3035849

[15] Y. Gao, T. Tang, S. Sun, Y. Wu, and P. Wang, "Digital twin-based office equipment management and personnel detection system," in *2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2024: IEEE, pp. 1651-1656.

[16] S. Khan, A. Alzaabi, Z. Iqbal, T. Ratnarajah, and T. Arslan, "A Novel Digital Twin (DT) model based on WiFi CSI, Signal Processing and Machine Learning for patient respiration monitoring and decision-support," *IEEE Access*, vol. 11, pp. 103554–103568, 2023. https://doi.org/10.1109/access.2023.3316508

[17] T. Fei *et al.*, "Maketwin: A reference architecture for digital twin software platform," *Chinese Journal of Aeronautics*, vol. 37, no. 1, pp. 1-18, 2024. https://doi.org/10.1016/j.cja.2023.05.002

[18] A. Freier, P. Karlton, and P. Kocher, "The secure sockets layer (SSL) protocol version 3.0," *RFC Editor*, 2011. https://doi.org/10.17487/RFC6101

[19] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," *RFC Editor*, 2018. https://doi.org/10.17487/rfc8446

[20] MQTT Version 3.1.1, "MQTT version 3.1.1 plus Errata 01," Retrieved: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html 2024.

[21] C. B. Gemirter, C. Senturca, and S. Baydere, "A comparative evaluation of AMQP, MQTT and HTTP protocols using real-time public smart City data," presented at the 6th International Conference on Computer Science and Engineering (UBMK), pp. 542–547, 2021.

[22] M. Ahmed and M. M. Akhtar, "Smart home: Application using HTTP and MQTT as communication protocols," *arXiv preprint arXiv:2112.10339*, 2021. https://doi.org/10.48550/ARXIV.2112.10339

[23] D. Skvorc, M. Horvat, and S. Srbljic, "Performance evaluation of Websocket protocol for implementation of full-duplex web streams," in *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2014: IEEE, pp. 1003-1008.

[24] L. P. Nam, T. N. Cat, D. T. Nam, N. Van Trong, T. N. Le, and C. Pham-Quoc, "OPC-UA/MQTT-based multi M2M protocol architecture for digital twin systems," in *International Conference on Intelligence of Things*, 2023: Springer, pp. 322-338.

[25] A. Kaufmann, T. Stamler, S. Peter, N. K. Sharma, A. Krishnamurthy, and T. Anderson, "TAS: TCP acceleration as an OS service," in *Proceedings of the Fourteenth EuroSys Conference 2019. https://doi.org/10.1145/3302424.3303985*, 2019, pp. 1-16.

[26] R. Fielding and J. Reschke, "Hypertext transfer protocol (HTTP/1.1): Semantics and content," 2014. https://doi.org/10.17487/rfc7231

[27] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power CoAP for Contiki," in *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, 2011: IEEE, pp. 855-860.

[28] F. Tao and Q. Qi, "Make more digital twins," *Nature*, vol. 573, no. 7775, pp. 490-491, 2019. https://doi.org/10.1038/d41586-019-02849-1

[29] S. S. Nair, "Securing against advanced cyber threats: A comprehensive guide to phishing, XSS, and SQL injection defense," *Journal of Computer Science and Technology Studies*, vol. 6, no. 1, pp. 76-93, 2024. https://doi.org/10.32996/jcsts.2024.6.1.9

[30] G. Pellegrino, M. Johns, S. Koch, M. Backes, and C. Rossow, "Deemon: Detecting CSRF with dynamic analysis and property graphs," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1757-1771.