ITAI 3377

Professor Patricia McManus

Hoang Dinh

Reflective Journal: IIoT Protocols Project

# Introduction

The IIoT Sensor Network Simulation is a project that shows how to collect, send, and display sensor data in real-time. It uses three communication methods—MQTT, CoAP, and OPC UA—to send fake temperature and humidity data from virtual sensors, similar to what happens in real factories. MQTT works with a publish-subscribe setup, CoAP follows a simple web-based approach, and OPC UA handles larger-scale communication. The data is shown live using a Matplotlib script, making it easy to watch how the sensors behave. There are also saved graphs that show earlier data. This project is a starting point for Industrial IoT (IIoT) projects and can grow to include cloud services, AI-based maintenance, or database storage.

The IIoT Sensor Network Simulation project helps me learn more about how industrial communication methods work, how to process data in real-time, and how to display it visually. My main goal is to get practical experience with MQTT, CoAP, and OPC UA, understanding how each one works in IoT systems and where they are most useful. By working on real-time data collection and visualization, I want to build solid skills in managing sensor data, which is important for automation and smart technologies.

This project also helps me sharpen my Python skills, especially in managing asynchronous tasks, creating visualizations with Matplotlib, and working with various network setups. I aim to improve my ability to design scalable and efficient systems for smooth data flow and real-time monitoring across different communication methods.

This project is not only technical learning for me but also a starting point for connecting IIoT systems with cloud platforms, databases, and AI-driven tools in the future. I want to explore how these technologies are used in real-world industries and possibly grow the project to include machine learning for predicting maintenance issues or saving data in an organized database for long-term study. In the end, I hope this project will sharpen my problem-solving skills, expand my understanding of IoT systems, and get me ready for future work in automation, autonomous systems, and smart IoT solutions.

# Personal Contributions

I independently completed every aspect of this project, assuming it was an individual assignment and not a team effort. From setting up the IIoT protocols (MQTT, CoAP, and OPC UA) to troubleshooting network issues, handling real-time data visualization, and implementing security measures, I tackled every challenge on my own. I spent a significant amount of time debugging errors, optimizing system performance, and ensuring proper integration of all components without relying on anyone else. Resolving issues such as connection failures, protocol mismatches, and visualization problems required extensive research, testing, and iteration. Although this process was time-consuming, it allowed me to develop a deep understanding of industrial IoT protocols and real-world problem-solving skills.

# Learning Outcomes

This project helped me thoroughly understand MQTT, CoAP, and OPC UA, which are key communication methods in Industrial IoT (IIoT). Each has its own role and is tailored to make data transfer efficient in various industrial settings. By working on real-time data transmission and visualization, I got to learn about their designs, advantages, and weaknesses.

MQTT is a simple messaging protocol that works well in networks with low bandwidth and high delays. It uses a central system called a broker, where devices send messages to specific topics. Other devices subscribed to those topics receive updates when new data comes in. This makes MQTT great for IoT systems that need to share data in real time across multiple devices. I learned how different Quality of Service (QoS) levels affect message delivery: QoS 0 delivers messages at most once, QoS 1 ensures at least one delivery, and QoS 2 guarantees exactly one delivery. Retained messages and Last Will and Testament (LWT) messages also help make communication more reliable, especially if networks get disconnected. However, MQTT’s reliance on a central broker can be a weak point unless managed carefully. While it doesn't have built-in security, you can add TLS encryption and authentication to make it secure.

CoAP is a request-response protocol made for devices with low power and limited resources. Unlike MQTT, which uses TCP and a central broker, CoAP runs on UDP, making it faster and better for small IoT devices. Its design is similar to HTTP, so it uses standard actions like GET, POST, PUT, and DELETE to work with sensor data. One useful feature I learned about is CoAP's ability to observe resources, where a client can get real-time updates without checking repeatedly, saving network usage. CoAP also supports multicast, so multiple devices can receive the same message efficiently. However, since UDP doesn’t guarantee message delivery, CoAP is less reliable than MQTT or OPC UA unless you add DTLS encryption for security. It works best for things like environmental monitoring, smart cities, and sensor networks where saving power is important.

OPC UA is a client-server protocol designed for secure and scalable communication in industrial systems. Unlike MQTT and CoAP, which are simpler messaging methods, OPC UA uses a structured data model to organize variables, objects, and methods in a clear hierarchy. This makes it perfect for smart factories, SCADA systems, and predictive maintenance. One key feature I learned is that OPC UA has built-in security features like encryption, authentication, and access control to ensure safe data sharing. It also uses a subscription system, so clients get real-time updates without constantly asking for data, reducing network traffic and improving efficiency. However, OPC UA needs more resources than MQTT and CoAP, which makes it less ideal for battery-powered devices or limited networks. It is more complex to learn but stands out for its ability to work across platforms and support communication between different vendors, making it a strong choice for IIoT projects.

When comparing these three protocols, I found that MQTT works best for large, scalable IoT systems, CoAP is great for low-power and limited networks, and OPC UA is the strongest option for industrial automation. MQTT is flexible and reliable for sharing messages, CoAP is very efficient for small, power-saving devices, and OPC UA provides secure and structured communication for industrial systems. The right choice depends on things like speed, security, energy use, and scalability. MQTT is simple and widely used, CoAP is better for saving energy, and OPC UA is vital for complex industrial communication.

One of the most important insights I gained was the importance of selecting the right protocol based on the use case and network constraints. MQTT, with its publish-subscribe setup, worked well for real-time IoT tasks needing constant data sharing, like remote monitoring. However, using a central broker caused delays in very busy networks. CoAP, being lightweight and using a request-response system, was great for low-power devices but wasn’t very reliable in networks with a lot of data loss because it runs on UDP. OPC UA stood out for its secure, structured, and consistent data exchange, making it perfect for industrial automation, but it required more computing power than MQTT and CoAP.

Another key takeaway was the practical challenges in real-time data visualization. Initially, I faced problems with Matplotlib freezing, especially when working with asynchronous event loops in CoAP and OPC UA. Through this experience, I realized how crucial multi-threading and event-driven programming are. I solved the issue by running Matplotlib in a separate thread, which kept the user interface working smoothly. I also learned how important it is to synchronize data from different sensors to ensure the graphs were accurate, properly aligned in time, and updated efficiently.

From a system design point of view, I understood how important error handling and debugging are in networked applications. For instance, in CoAP, I faced "4.05 Method Not Allowed" errors, which required changing the server to accept POST requests correctly. In OPC UA, managing node subscriptions efficiently was key to reducing network traffic and avoiding delays. These challenges taught me better debugging methods, like keeping logs of incoming data, using protocol analyzers to check communication, and creating backup plans for unreliable networks.

Security was another important area I focused on. MQTT and CoAP need extra security measures like TLS and DTLS, but OPC UA has built-in authentication and encryption, which makes it a better choice for critical industrial tasks. This showed me the balance needed between security and performance, as stronger encryption can slow things down and add more processing work.

This project not only boosted my technical knowledge but also improved my problem-solving and adaptability. Each protocol came with its own challenges—like MQTT's reliance on a central broker, CoAP's unreliable data transmission, and OPC UA's complexity. Tackling these issues helped me develop a clear and organized way to troubleshoot, making me better at identifying and fixing problems in real-world industrial systems.

# Challenges and Outcomes

One of the first challenges I faced was establishing a stable MQTT connection between the sensor simulation script and the broker. Sometimes, the client couldn’t connect, or the subscriber didn’t receive messages properly. This happened because the broker wasn’t running as it should, and the message retention settings weren’t consistent. To fix this, I made sure the MQTT broker (Mosquitto) was always active before starting the sensor simulation. I also enabled message retention and used QoS (Quality of Service) levels to make communication more reliable. The main takeaway for me was that proper broker management is key for MQTT, and features like QoS, retained messages, and reconnection strategies are essential for keeping IoT communication stable and scalable.

When working with CoAP, I ran into a "4.05 Method Not Allowed" error while trying to send sensor data using POST requests. This happened because CoAP servers usually only handle GET requests unless you set them up to accept POST requests. To fix this, I adjusted the CoAP server to allow POST requests, so it could properly receive and store sensor data. I also added error handling to return clear response codes when an invalid request was made. This experience showed me that CoAP needs thoughtful API design since it uses a request-response model, unlike MQTT's publish-subscribe system. It highlighted the importance of defining clear endpoints and managing different request types in IoT applications.

A big challenge I faced was real-time data visualization with Matplotlib while fetching data asynchronously from different protocols. At first, Matplotlib would freeze or crash, especially when I used plt.pause() in an asyncio event loop. This happened because Matplotlib’s GUI functions were blocking the asynchronous loop, stopping real-time data processing. To fix this, I ran Matplotlib in a separate thread to keep the UI responsive while the sensors kept sending data. I also replaced plt.pause() with plt.draw() to update the visuals without blocking the program. The key takeaway was that combining real-time visualization with async communication needs careful thread management to keep everything running smoothly.

When working with OPC UA, I faced challenges with data subscriptions and node addressing. At first, the client couldn't retrieve sensor values because the namespace indexing and node paths were incorrect. To fix this, I used an OPC UA browser to explore the namespaces and correctly identify the node paths. I also switched from polling to using OPC UA subscriptions, which reduced extra network traffic and made the process more efficient. This experience taught me that OPC UA is different from MQTT and CoAP—it doesn't rely on simple message exchanges but instead uses structured data models with hierarchical nodes. Correctly addressing nodes and using subscriptions instead of polling is essential for effective real-time industrial monitoring.

One of the more complex challenges was synchronizing sensor data from multiple protocols in a single visualization. Since MQTT, CoAP, and OPC UA work separately, their data would arrive at different times, causing mismatched values in the graph. To fix this, I added timestamps (using datetime.now()) to each data point to align them properly. I also created data buffers for each protocol to store the last received values and avoid gaps in the graph. Another key lesson was handling missing data effectively—by either filling in gaps between known values or using placeholders, I made sure the graph stayed clear and helpful.

Lastly, I encountered important security concerns when dealing with real-time industrial data transmission. By default, MQTT, CoAP, and OPC UA don’t include encryption, leaving sensor data open to interception. To fix this, I added TLS encryption for MQTT, DTLS for CoAP, and used OPC UA’s built-in authentication and security features. This taught me that security needs to be a priority in IIoT applications. While MQTT and CoAP need extra security measures, OPC UA stands out as the most secure option right from the start, making it a great choice for critical industrial tasks.

Facing these challenges gave me a deeper understanding of picking the right protocol, ensuring reliable communication, and creating real-time data visuals. I learned that MQTT is very scalable but needs good broker management, CoAP works well for limited devices but demands careful API planning, and OPC UA is strong for industrial automation but takes more effort to learn. Along the way, I also sharpened my problem-solving skills, like fixing protocol errors, improving real-time data graphs, and applying security best practices.

This project highlighted how important it is to manage asynchronous communication well, keep real-time visualizations responsive, and align data from different sources. In the future, I plan to use these lessons for bigger IIoT projects, like connecting to cloud-based IoT platforms, using AI for predictive maintenance, and storing sensor data in databases for long-term analysis.

# Future Applications

The knowledge I gained from the IIoT Sensor Network Simulation project can be directly used in future IoT and industrial automation projects, especially for real-time data processing, combining protocols, and designing systems that can scale. By understanding the pros and cons of MQTT, CoAP, and OPC UA, I can choose the best communication protocol for specific IoT applications—like MQTT for scalable cloud monitoring, CoAP for low-power devices, and OPC UA for secure industrial automation. Debugging network issues, managing asynchronous data visualization, and adding security measures have strengthened my ability to build reliable and efficient IIoT systems. These skills are especially useful in areas like smart manufacturing, predictive maintenance, and AI-driven industrial systems, where real-time sensor data is essential. My experience with integrating multiple protocols into one visualization system can also lead to creating cloud-connected dashboards, database analytics, and AI tools for detecting issues. This project has built a strong foundation in IIoT technologies and prepared me for future roles in IoT development, automation, and smart system design.

Possible improvements for this project include adding a database system like MySQL, PostgreSQL, or InfluxDB to save sensor data over time, making it easier to analyze trends and monitor long-term activity. Another upgrade could be connecting to cloud platforms like AWS IoT, Azure IoT Hub, or Google Cloud IoT to allow remote access and advanced data analysis. Using machine learning models for predictive maintenance could also help spot issues in sensor readings early, boosting efficiency and reducing downtime. Creating a web-based dashboard with tools like Flask, Django, or Node.js and React could make it easier to view real-time data across devices. Lastly, strengthening security by adding end-to-end encryption, authentication, and role-based access control would make the system more reliable for use in smart factories and industrial automation.