Eduardo Cabrera-lopez

ITAI 3377

June 17, 2025

IIoT Protocols Project Introduction Reflective Journal

This research was meant to enable me to investigate and grasp in a simulated edge-computing environment three fundamental Industrial Internet of Things (IIoT) communication protocols—MQTT, Coap, and OPC UA—operate. Along with following each protocol end-to-end, my own objective was to personally see the difficulties with real-time data flow, processing, and visualization. Simulating temperature and humidity sensor data, then forwarding it via several protocols, would help me to appreciate the trade-offs each protocol provides in terms of dependability, overhead, and simplicity of integration.

Individual Contributions

Every facet of the code development and environment configuration fell to me:

I installed and setup the Mosquitto broker locally, then built mqtt_sensor_simulation.py using the paho-mqtt library to post randomized sensor data every second.

After trying the original aiocoap client/server technique and running across ongoing socket-binding problems on Windows, I decided to create a fallback utilizing raw UDP sockets, therefore keeping the CoAP request/response pattern and guaranteeing dependable local testing.

I set up an OPC UA server at opc.tcp://0.0.0.0:4840/freeopcua/server/. Using the asyncua library I modeled two variables—temperature and humidity—exposed them as editable nodes and created an asynchronous loop to continually update them.

I developed data_visualization.py with Matplotlib and Pandas to listen to the MQTT topic, parse arriving JSON payloads, and plot temperature and humidity against a real-time timestamp axis. I turned several charting errors around to get a neat live chart.

Along with this reflection notebook, I arranged the folder structure, created a human-friendly README, and developed a 1,000-word comparison report to help to consolidate lessons acquired and future directions.

MQTT: I discovered how dependable, ordered, low-latency messages might be produced using a lightweight, TCP-based publish/subscribed paradigm. Dealing with Mosquitto and paho-mqtt helped one to understand why MQTT is the preferred method for telemetry in limited IoT systems.

Although CoAP's UDP basis makes it best for lossy or low-power situations, its conceptually like REST over HTTP. Real-world library support can lag, too; on Windows, aiocoap displayed socket binding problems that needed me to grasp OS-level networking to address.

OPC UA: I came across the value of an industrial automation rich, object-oriented data model. Exposing structured nodes and viewing them with OPC UA clients first caught my attention through the asyncua API. Though I delayed complete TLS implementation, I personally experienced how OPC UA enables strong security and metadata formats.

Debugging Matplotlib's default date parsing taught me how to preprocess timestamps into string labels and manually regulate axis limits—skills I'll carry into future real-time data projects. For on-demand streamlining of data, Pandas' DataFrame interface proved quite helpful.

Problems and Remarks on Their Solutions

Repeated [WinError 10049] difficulties binding the CoAP server prompted me to pivot. Without depending on a non-functional library on my platform, I investigated asynchronous UDP in Python, built a basic UDP client/server pair, and kept the CoAP ethos.

X-Axis Date Misinterpretation: Originally shown years in the future, the live MQTT plot I figured this out simply substituting datetime. now() with string-formatted HH:MM:SS labels, then rebuilt appropriate Pandas datetime parsing alongside explicit plt.xlim() calls to zoom into the current window.

Using asyncio came from learning from ensuring clean startup and shutdown of asynchronous servers (Coap and OPC UA). run() patterns and context managers like async with server to prevent resource leakage.

My confidence in reading stack traces, tracking socket-level activities, and reevaluating design grew through every obstacle. To direct my repairs, I turned to official documentation, Stack Overflow, and GitHub problems.

Future Uses

These events have helped me to feel ready to:

Using TLS for both MQTT and OPC UA channels, apply secure telemetry using certificates and mutual authentication.

Create online dashboards visualizing sensor networks in the browser using frameworks like Flask or Streamlit, maybe aggregating several protocol streams into a single UI.

Using technologies like InfluxDB or Grafana for long-term storage and analysis, scale to multi-sensor setups whereby hundreds of simulated nodes submit data concurrently. Apply to actual smart manufacturing or building automation projects where MQTT pipelines feed AI-driven anomaly detection at the edge and OPC UA metadata can interface with enterprise systems.

In conclusion, completing Lab 04 has given me not just code samples but also architectural insights and practical debugging techniques that will guide my work as an AI engineer in industrial and IoT spheres.