

Week 3

A – Class Discussion

Mapping IoT Communications to the OSI Model

- How does IoT communications map to the Physical/Data Link Layer of the OSI Model?
 - At the Physical and Data Link layers, IoT communications involve the actual transmission of data over a physical device.
 - At Physical Level – this layer is concerning on how each physical device is connecting (physically) to the network with hardware – e.g: cable, wire, radio, wireless network.
 - At Data Level - this layer is responsible for framing data, physical addressing (MAC addresses), error detection, and flow control within a local network.
 - MAC protocols (e.g., CSMA/CA for Wi-Fi, TDMA/CSMA for Zigbee): Manage access to the shared physical medium.
 - Error detection and correction mechanisms: Ensure data integrity.
 - Framing: Structuring data into frames for transmission.
- How does IoT communications map to the Network/Transport Link Layer of the OSI Model?
 - These two layers are responsible for enabling end-to-end communication across different networks and ensuring reliable data delivery.
 - Network Layer: This layer is responsible for logical addressing (IP addresses), routing data packets across different networks, and determining the best path for data.
 - IPv6/IPv4: IPv6 is becoming increasingly important for IoT due to the massive number of devices requiring unique addresses.
 - IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN): A key adaptation layer that allows IPv6 packets to be sent over low-power wireless networks like IEEE 802.15.4, making it possible for resource-constrained IoT devices to participate in IP-based networks.

➤ Routing Protocols:

- ❖ RPL (Routing Protocol for Low-Power and Lossy Networks): A routing protocol specifically designed for resource-constrained IoT devices and mesh networks, often used with 6LoWPAN.

- Transport Layer: This layer provides end-to-end communication between applications, ensuring reliable data transfer and flow control.

- UDP (User Datagram Protocol): Often preferred in IoT for its low overhead, making it suitable for devices with limited resources and applications where occasional data loss is acceptable
- TCP (Transmission Control Protocol): Used when reliable, ordered, and error-checked data delivery is critical (e.g., firmware updates, critical control commands).
- DCCP (Datagram Congestion Control Protocol): Less common in general IoT but can be used for connection-oriented, unreliable datagram flow with congestion control.

- How does IoT communications map to the Session/Presentation/Application Layer of the OSI Model?

- These three layers are dealing with inter-application communication, data formatting, and providing services directly to the end-user or other applications.

- Session Layer – establishes, manages, and terminates communications sessions between applications.
- Presentation Layer – responsible for data formatting, encryption, decryption, and compression, ensuring that data is presented in a readable format for the application layer.
- Application Layer – directly interacts with software applications. It provides high-level services and protocols for specific IoT use cases

Low Power Communications

- Why do we need low power communications for IoT?
 - Extended Battery Life:
 - Most IoT devices run on batteries.
 - Low-power communication allows them to operate for months or even years without needing battery changes → cutting down on maintenance costs and effort, especially for widespread deployments.
 - Scalability:
 - To manage billions of connected devices, each one must be energy efficient.
 - Powering and maintaining a massive network of power-hungry devices would be logistically impossible and financially unsustainable.
 - Deployment Flexibility:
 - Low-power devices can be placed almost anywhere, including remote or hard-to-access locations without traditional power sources, broadening the scope of IoT applications.
 - Cost-Effectiveness:
 - Less power consumption means simpler, cheaper components and smaller batteries, which lowers the overall manufacturing cost of devices and reduces long-term operational expenses.
 - Miniaturization:
 - Lower power requirements mean less heat generation and smaller power components, enabling more compact and integrated device designs suitable for small products or wearables.
- What are the characteristics of LoWPAN technologies?
 - Concept: Localized, short-range, low-power networks, often for personal or small-area use. Think of devices talking within a room or building.

- Range: Short (tens of meters). Ideal for personal area networks (PANs) or small-scale local deployments.
 - Power Consumption: Low. Designed for battery life from months to a few years.
 - Data Rates: Low. Suitable for small, infrequent data packets (e.g., sensor readings, simple commands). Not for streaming.
 - Networking: Often supports mesh networking, where devices can relay messages for neighbors, extending local reach and improving resilience.
 - IP Integration: Crucially, often supports IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN), allowing resource-constrained devices to participate directly in IP networks.
 - Cost: Generally low cost per node and relatively simple infrastructure.
 - Examples: Zigbee, Bluetooth Low Energy (BLE), Thread.
- What are the characteristics of LPWAN technologies?
 - Concept: Localized, short-range, low-power networks, often for personal or small-area use. Think of devices talking within a room or building.
 - Range: Short (tens of meters). Ideal for personal area networks (PANs) or small-scale local deployments.
 - Power Consumption: Low. Designed for battery life from months to a few years.
 - Data Rates: Low. Suitable for small, infrequent data packets (e.g., sensor readings, simple commands). Not for streaming.
 - Networking: Often supports mesh networking, where devices can relay messages for neighbors, extending local reach and improving resilience.
 - IP Integration: Crucially, often supports IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN), allowing resource-constrained devices to participate directly in IP networks.
 - Cost: Generally low cost per node and relatively simple infrastructure.
 - Examples: Zigbee, Bluetooth Low Energy (BLE), Thread.

Application Layer Communications

- What are the characteristics of RESTful communication?
 - REST, fundamentally an architectural style often implemented over HTTP/HTTPS, offers a familiar paradigm for web developers.
 - **Statelessness:**
 - Each client request is self-contained.
 - The server doesn't retain session information between requests.
 - While excellent for scalability (any server can handle any request), it can add overhead for repetitive interactions, which might be a concern for resource-constrained IoT devices or very frequent data points.
 - **Client-Server Decoupling:**
 - A clear separation, allowing independent evolution of the client (device) and server (cloud platform).
 - This is beneficial for large, distributed systems.
 - **Cache-ability:**
 - Responses can be marked as cacheable, reducing network traffic and server load for frequently accessed, unchanging data.
 - This can save bandwidth for devices if used correctly.
 - **Uniform Interface:**
 - Relies on standard HTTP methods (GET, POST, PUT, DELETE) to interact with identified resources (via URIs).
 - This consistency aids development but might be too verbose for extremely constrained devices.
 - **Synchronous by Nature:**
 - Typically, a client sends a request and waits for a response.

- This can introduce latency and isn't ideal for event-driven, push-style communication common in sensor networks.
- What are the characteristics of MQTT communication?
 - MQTT is purpose-built for constrained environments and messaging patterns common in IoT.
 - Publish/Subscribe Model:
 - Clients publish messages to abstract "topics," and other clients subscribe to those topics.
 - A central broker handles message routing.
 - This inherently decouples senders from receivers, improving scalability and flexibility.
 - Lightweight and Efficient:
 - Designed with minimal overhead in mind, featuring compact message headers and a small code footprint.
 - This makes it highly efficient for constrained devices and low-bandwidth networks.
 - Asynchronous Communication:
 - Publishers and subscribers don't need direct knowledge of each other or simultaneous online presence.
 - The broker queues for messages.
 - This is crucial for devices with intermittent connectivity (e.g., LPWAN devices that "wake up, send, and sleep").
 - Quality of Service (QoS):
 - Offers distinct levels (0, 1, 2) for message delivery guarantees, allowing developers to balance reliability with overhead based on data criticality.
 - Persistent Sessions & Last Will:

- Clients can maintain session state with the broker, ensuring messages are delivered even after a temporary disconnection.
 - The "Last Will and Testament" feature provides critical device status notification upon unexpected disconnections.
- What do we need from an application layer communications protocol to build a scalable IoT application?
 - Efficiency for Constrained Devices: Protocols must be extremely lightweight in terms of payload size, header overhead, and computational demands (CPU, RAM). This directly impacts battery life and hardware cost.
 - Network Resilience: Must perform well over unreliable, low-bandwidth, and high-latency networks (e.g., LPWANs). Features like asynchronous operation, message queuing, and QoS are critical.
 - Massive Scalability: The architecture needs to gracefully handle millions to billions of concurrently connected devices and trillions of messages. This often points towards decoupled, broker-centric models (like pub/sub).
 - Security Integration: Robust security mechanisms (encryption, authentication, authorization) are non-negotiable for data integrity and device security. This includes integration with transport layer security (TLS/DTLS).
 - Bidirectional Communication: While many IoT devices are telemetry producers, the ability to receive commands, firmware updates, or configuration changes from the cloud is essential for management and control.
 - Interoperability: Adherence to open standards is vital to prevent vendor lock-in and enable heterogeneous device ecosystems.
 - Support for Diverse Data Models: Flexibility to handle various data formats (e.g., JSON, CBOR, Protobuf) efficiently, allowing developers to choose the most suitable one for their payload size and parsing needs.

- **Device Management Capabilities:** While not always solely at the application layer, protocols that facilitate remote provisioning, monitoring, and updating devices (e.g., LwM2M often layered on CoAP) are highly advantageous for scalability.

B – Group Discussions

For the application `Vehicle Asset Health` identified in the image.

1. What functionality is needed for this application? What should be sensed, what processing is needed and what actuation is possible?
 - **Functionality:** this application must be able to detect the current “health status” of a vehicle based on the following factors:
 - **Vehicle parameters:**
 - Engine temperature.
 - Oil pressure.
 - Fuel levels.
 - Tire pressure.
 - Brake wear.
 - **Operational data:**
 - Speed.
 - Runtime hours.
 - Load weight.
 - **Driver behaviours:**
 - Acceleration patterns.
 - Braking frequency.
 - **Processed:**
 - **Data Analysis:** Real-time analysis to detect anomalies (e.g., engine overheating, abnormal vibrations).
 - **Predictive Maintenance:** Algorithms to predict when maintenance is needed based on sensor data trends.
 - **Diagnostics:** Fault code generation and prioritization (e.g., critical vs. non-critical issues).
 - **Aggregation:** Combining data from multiple vehicles for fleet-wide insights.
 - **Alerts:** Generating notifications for maintenance teams or operators when thresholds are exceeded.
 - **Actuated:**

- **Alerts/Notifications:** Sending warnings to drivers (e.g., dashboard alerts) or maintenance crews (e.g., via mobile apps or control centers).
- **Automated Controls:** Shutting down a vehicle in critical failure scenarios or limiting speed if unsafe conditions are detected.
- **Maintenance Scheduling:** Automatically logging maintenance requests in a central system.
- **Reporting:** Generating reports for fleet managers on vehicle health status.

2. What web services would you need to structure to implement this?

- **Data Collection Service:**
 - Purpose: Collects sensor data (e.g., engine temperature, GPS) from vehicles.
 - Implementation: RESTful API (e.g., POST /vehicles/{vehicle_id}/sensors) to receive data from IoT devices on vehicles.
 - Example: Vehicles send JSON payloads with sensor readings to a cloud endpoint.
- **Data Processing Service:**
 - Purpose: Processes raw sensor data for anomaly detection and predictive maintenance.
 - Implementation: A microservice that consumes sensor data, runs analytics (e.g., machine learning models), and stores results in a database.
 - Example: POST /analytics/process to trigger analysis and return diagnostic results.
- **Notification Service:**
 - Purpose: Sends alerts to drivers, maintenance teams, or fleet managers.
 - Implementation: REST API or WebSocket for real-time notifications (e.g., POST /notifications/alert to send emails, SMS, or app notifications).
 - Example: Alerts for critical issues like engine failure.
- **Maintenance Scheduling Service:**
 - Purpose: Logs and schedules maintenance tasks based on processed data.
 - Implementation: REST API (e.g., POST /maintenance/schedule) integrated with a maintenance management system.
 - Example: Creates a work order for a vehicle needing tire replacement.
- **Dashboard Service:**
 - Purpose: Provides a user interface for fleet managers to monitor vehicle health.
 - Implementation: GET /dashboard/vehicles to retrieve real-time and historical data for visualization.
 - Example: Displays vehicle status, alerts, and maintenance schedules.

- Authentication/Authorization Service:
 - Purpose: Secures access to services and data.
 - Implementation: OAuth 2.0-based API (e.g., POST /auth/token) to authenticate users and devices.
 - Example: Ensures only authorized personnel access sensitive vehicle data.
- Assumptions: Services are cloud-based, scalable, and use standard protocols like HTTPS for security.

3. What hierarchical MQTT topics would you use?

- For Vehicle Asset Health, topics should reflect the mining site, vehicle, and data type.
- Possible topic structure:

“mining/{site_id}/{vehicle_id}/{data_type}/{sensor_type}”
- Where:
 - site_id: Unique identifier for the mining site (e.g., siteA).
 - vehicle_id: Unique vehicle identifier (e.g., truck001).
 - data_type: Category of data (e.g., sensors, alerts, commands).
 - sensor_type: Specific sensor or action (e.g., engine_temp, tire_pressure, maintenance).
- Example Topics:
 - **Sensor Data:**
 - mining/siteA/truck001/sensors/engine_temp: Publishes engine temperature readings.
 - mining/siteA/truck001/sensors/tire_pressure: Publishes tire pressure data.
 - **Alerts:**
 - mining/siteA/truck001/alerts/critical: Publishes critical alerts (e.g., engine failure).
 - mining/siteA/truck001/alerts/warning: Publishes non-critical warnings (e.g., low fuel).
 - **Commands:**
 - mining/siteA/truck001/commands/shutdown: Sends a command to shut down the vehicle.
 - mining/siteA/truck001/commands/schedule_maintenance: Triggers a maintenance request.
- Wildcards for Filtering:
 - **Single-level (+):** Subscribe to mining/siteA/+/sensors/engine_temp to receive engine temperature data from all vehicles at siteA.

- **Multi-level (#):** Subscribe to mining/siteA/truck001/# to receive all messages for truck001 (sensors, alerts, commands).
- **Fleet-wide Monitoring:** Subscribe to mining/+/+/sensors/# to monitor sensor data across all sites and vehicles.

Based on this, continue with the following.

4. Create a high-level block diagram of the scenario, include the all the people, places and things that are involved in the scenario.

Decide what wireless communications technology to use.

5. Create a simple data flow diagram showing all the needed communication in this system.

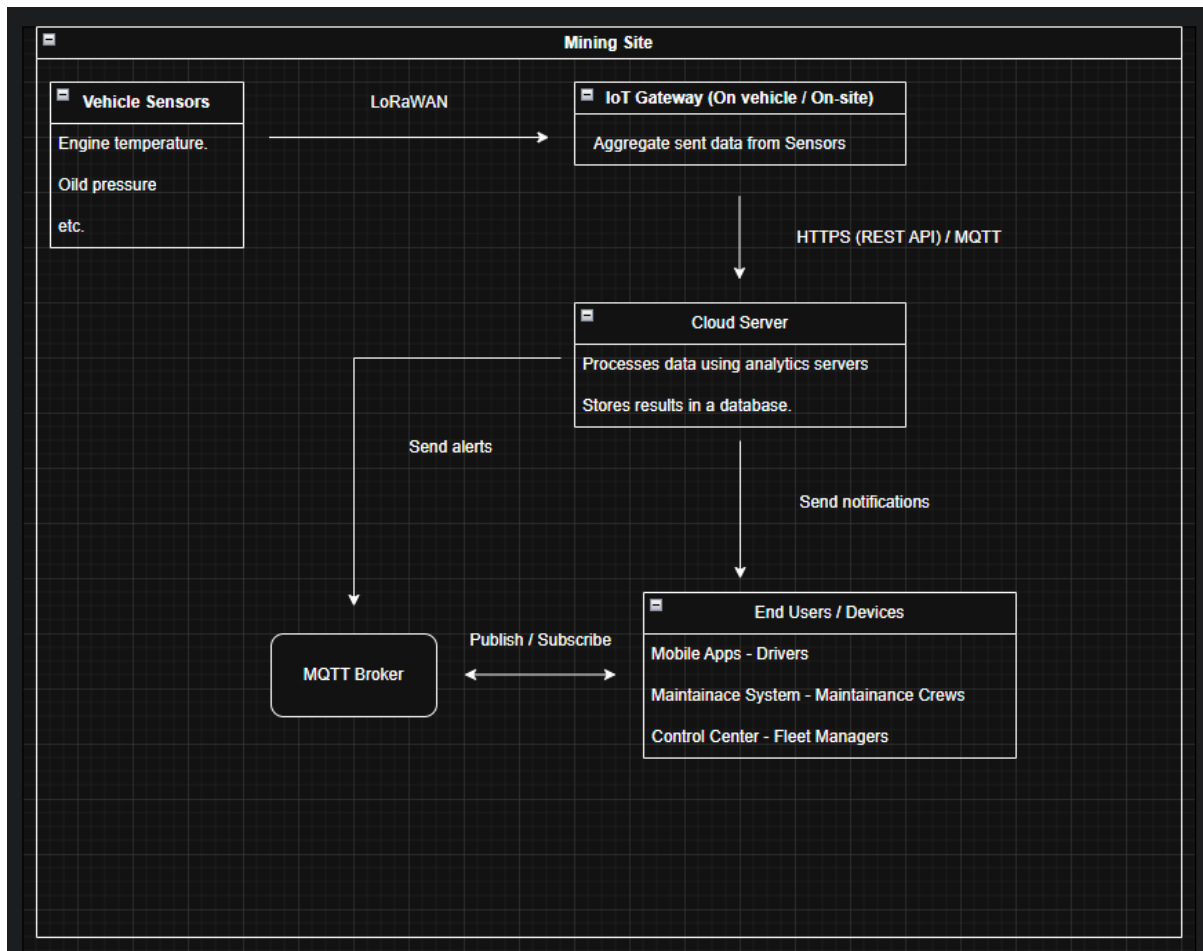


Diagram 1 – Flow diagram for both question 4 & 5.

Design two approaches to this application:

6. Plan a web service-based approach to implementing the application-level communications for this application. What services would you need?

- **Services Needed**

- **Sensor Data Ingestion:**

- **Endpoint:** POST /vehicles/{vehicle_id}/sensors
- **Function:** Receives JSON payloads with sensor data (e.g., { "engine_temp": 85, "tire_pressure": 30 }).
- **Security:** Authenticated via OAuth 2.0 device tokens.

- **Analytics Service:**

- **Endpoint:** POST /analytics/process/{vehicle_id}

- **Function:** Runs predictive maintenance algorithms and generates diagnostics.
 - **Output:** Stores results in a database and triggers alerts if needed.
- **Alert Service:**
 - **Endpoint:** POST /notifications/alert
 - **Function:** Sends notifications via email, SMS, or push notifications to users.
 - **Example:** { "vehicle_id": "truck001", "message": "Engine overheating", "priority": "critical" }.
- **Maintenance Service:**
 - **Endpoint:** POST /maintenance/schedule
 - **Function:** Creates work orders in a maintenance system.
 - **Integration:** Connects to existing enterprise maintenance software.
- **Dashboard Service:**
 - **Endpoint:** GET /dashboard/vehicles/{site_id}
 - **Function:** Returns vehicle health data for visualization (e.g., JSON with status, alerts, and maintenance history).
- **Command Service:**
 - **Endpoint:** POST /vehicles/{vehicle_id}/commands
 - **Function:** Sends commands to vehicles (e.g., shutdown, speed limit).
 - **Security:** Restricted to authorized users (e.g., fleet managers).

- **Architecture:**

- **Microservices:** Each service is independently deployable, using a cloud platform like AWS or Azure.
- **Database:** A NoSQL database (e.g., MongoDB) for storing sensor data and analytics results.
- **API Gateway:** Manages routing, authentication, and rate limiting for all services.
- **Scalability:** Services scale horizontally to handle large fleets.
- **Security:** HTTPS, OAuth 2.0, and role-based access control (RBAC) for users and devices.

7. Plan a hierarchical MQTT application-level communications protocol. Think about the identity of users and devices in the system and plan single-level or multi-level wildcards for message filtering.

- **Identity Management:**

- **Devices:**
 - Each vehicle has a unique ID (e.g., truck001) embedded in MQTT topics.
 - IoT Gateways have IDs (e.g., gatewayA) for aggregating data from multiple vehicles.

- Devices authenticate with the MQTT broker using client certificates or username/password pairs.
- **Users:**
 - **Drivers:** Subscribe to vehicle-specific alerts (e.g., mining/siteA/truck001/alerts/#).
 - **Maintenance Crews:** Subscribe to site-wide alerts (e.g., mining/siteA+/alerts/#).
 - **Fleet Managers:** Subscribe to fleet-wide data (e.g., mining/+//sensors/#).
 - User authentication is handled via a separate authentication service (e.g., OAuth 2.0 tokens mapped to MQTT subscriptions).

- **Topic Hierarchy:**

“mining/{site_id}/{vehicle_id}/{data_type}/{sensor_type_or_action}”

- **Examples:**
 - Sensor data: mining/siteA/truck001/sensors/engine_temp
 - Alerts: mining/siteA/truck001/alerts/critical
 - Commands: mining/siteA/truck001/commands/shutdown
- **Gateway Topics:**
 - mining/siteA/gatewayA/sensors/aggregated: Publishes aggregated data from all vehicles at siteA.
- **Control Center Topics:**
 - mining/siteA/control_center/commands: Publishes commands to all vehicles at siteA.

- **Wildcard Usage:**

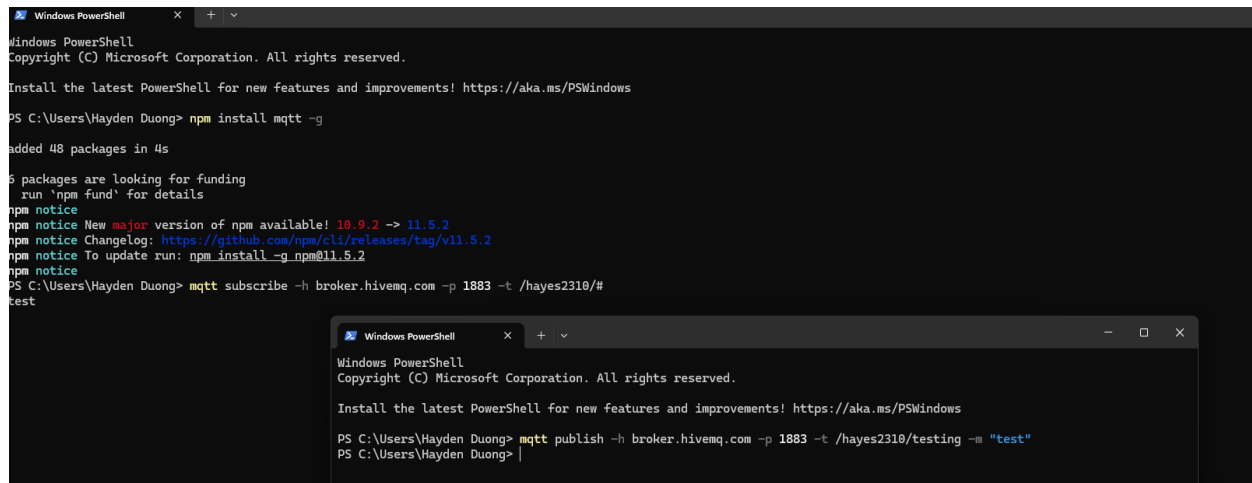
- **Single-level (+):**
 - Maintenance crews subscribe to mining/siteA+/alerts/critical to receive critical alerts from all vehicles at siteA.
 - Drivers subscribe to mining/siteA/truck001+/critical to receive critical alerts for their vehicle.
- **Multi-level (#):**
 - Fleet managers subscribe to mining/+//sensors/# to monitor all sensor data across all sites.
 - Control centers subscribe to mining/siteA/# for all messages from siteA.
- **Security Considerations:**
 - Use Access Control Lists (ACLs) in the MQTT broker to restrict subscriptions (e.g., drivers can only subscribe to their own vehicle’s topics).
 - Encrypt payloads with TLS to protect sensitive data.

- **Message Flow:**

- **Publish:** Vehicles publish sensor data to mining/siteA/truck001/sensors/*.
- **Subscribe:** Maintenance crews subscribe to mining/siteA+/alerts/# for real-time alerts.

- **Command:** Control centers publish to mining/siteA/truck001/commands/* to send instructions.
- **Aggregation:** Gateways publish aggregated data to mining/siteA/gatewayA/sensors/aggregated.

C – Technical Tasks



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Hayden Duong> npm install mqtt -g

added 48 packages in 4s

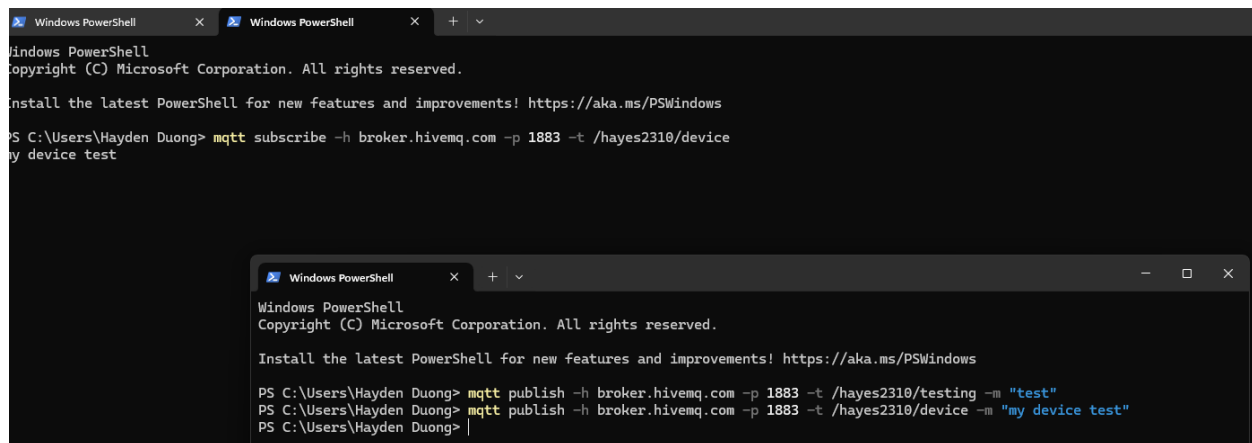
5 packages are looking for funding
  run 'npm fund' for details

npm notice
npm notice New major version of npm available! 10.9.2 -> 11.5.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.5.2
npm notice To update run: npm install -g npm@11.5.2
npm notice
PS C:\Users\Hayden Duong> mqtt subscribe -h broker.hivemq.com -p 1883 -t /hayes2310/#
test

PS C:\Users\Hayden Duong> mqtt publish -h broker.hivemq.com -p 1883 -t /hayes2310/testing -m "test"
PS C:\Users\Hayden Duong>

```

Picture 1 – Result gained from step 5 & 6.



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Hayden Duong> mqtt subscribe -h broker.hivemq.com -p 1883 -t /hayes2310/device
my device test

PS C:\Users\Hayden Duong> mqtt publish -h broker.hivemq.com -p 1883 -t /hayes2310/testing -m "test"
PS C:\Users\Hayden Duong> mqtt publish -h broker.hivemq.com -p 1883 -t /hayes2310/device -m "my device test"
PS C:\Users\Hayden Duong>

```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Hayden Duong> npm install mqtt -g

added 48 packages in 4s

6 packages are looking for funding
  run 'npm fund' for details
npm notice
npm notice New major version of npm available! 10.9.2 -> 11.5.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.5.2
npm notice To update run: npm install -g npm@11.5.2
npm notice
PS C:\Users\Hayden Duong> mqtt subscribe -h broker.hivemq.com -p 1883 -t /hayes2310/#
test
my device test
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Hayden Duong> mqtt publish -h broker.hivemq.com -p 1883 -t /hayes2310/testing -m "test"
PS C:\Users\Hayden Duong> mqtt publish -h broker.hivemq.com -p 1883 -t /hayes2310/device -m "my device test"
```

Picture 2 – Results gained from step 8 & 9 (both subscribers received “my device test”).

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Hayden Duong> npm install mqtt -g

added 48 packages in 4s

6 packages are looking for funding
  run 'npm fund' for details
npm notice
npm notice New major version of npm available! 10.9.2 -> 11.5.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.5.2
npm notice To update run: npm install -g npm@11.5.2
npm notice
PS C:\Users\Hayden Duong> mqtt subscribe -h broker.hivemq.com -p 1883 -t /hayes2310/#
test
my device test
my user test
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Hayden Duong> mqtt subscribe -h broker.hivemq.com -p 1883 -t /hayes2310/device
my device test
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Hayden Duong> mqtt publish -h broker.hivemq.com -p 1883 -t /hayes2310/testing -m "test"
PS C:\Users\Hayden Duong> mqtt publish -h broker.hivemq.com -p 1883 -t /hayes2310/device -m "my device test"
PS C:\Users\Hayden Duong> mqtt publish -h broker.hivemq.com -p 1883 -t /hayes2310/users -m "my user test"
```

Picture 3 – Result gained from step 10 (only one of the subscribers is received the message).


```
server.js
1 // server.js
2 // This code connects to an MQTT broker and listens for messages on a specific topic
3 const mqtt = require('mqtt');
4 const client = mqtt.connect('mqtt://broker.hivemq.com:1883');
5
6 // Define the topic to subscribe to
7 var topic = '/myid';
8
9 // Set up the event listener for when the client connects to the broker
10 // Subscribe to the topic and log messages received
11 client.on('connect', () => {
12   client.subscribe(topic);
13   console.log('mqtt connected');
14 });
15
16 client.on('message', (topic, message) => {
17   console.log('Topic is: ' + topic);
18   console.log('Message is: ' + message);
19 });
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
client.js
1 // client.js
2 // This code connects to an MQTT broker and publishes a message to a specific topic
3 // It uses the 'mqtt' library to handle the connection and publishing.
4 const mqtt = require('mqtt');
5 const client = mqtt.connect('mqtt://broker.hivemq.com:1883');
6
7 // Define the topic and message to be published
8 var topic = '/myid';
9 var message = 'My message';
10
11 // Set up the event listener for when the client connects to the broker
12 client.on('connect', () => {
13   console.log('mqtt connected');
14   client.publish(topic, message);
15   console.log('published to Topic: ' + topic + ' with Message: ' + message);
16 });
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
PS C:\Users\Wuyden Dang\Desktop\SIIT314_729 - Software Architecture and Scalability for IoT\Week 4 - Scalable of IoT Applications\Technical Tasks\week_3> node server.js
mqtt connected
Topic is: /myid
Message is: My message

PS C:\Users\Wuyden Dang\Desktop\SIIT314_729 - Software Architecture and Scalability for IoT\Week 4 - Scalable of IoT Applications\Technical Tasks\week_3> node client.js
mqtt connected
published to Topic: /myid with Message: My message
```

Picture 4 – Results gained from “Writing simple programs that use MQTT”.

Week 4

A – Class Discussion

Week 4 Class Discussion: Comparing IoT Development Platforms

I. Comparing IoT Development Platforms

A - Metrics for Comparing IoT Development Platforms

- Scalability: Ability to handle increasing numbers of devices and data volume.
- Device Management: Features for provisioning, monitoring, updating, and managing devices.
- Security: Protocols for authentication, encryption, and threat detection.
- Integration and Development: Support for SDKs, APIs, programming languages, and integration with other services.
- Analytics and Data Processing: Tools for real-time analytics, visualization, and machine learning.
- Cost Efficiency: Pricing models, including free tiers and per-device/message costs (e.g., AWS IoT Core free tier: 250,000 messages/month; Azure IoT Hub: 8,000 messages/day).

- **Ease of Use:** User-friendly interfaces, pre-built templates, and learning resources (e.g., Azure IoT Central's drag-and-drop tools, AWS tutorials).
- **Protocol Support:** Support for communication protocols like MQTT, HTTP, CoAP, or Zigbee (e.g., Node-RED supports MQTT, Zigbee2MQTT for Zigbee devices).
- **Edge Computing:** Capabilities for local data processing to reduce latency (e.g., AWS IoT Greengrass, Azure IoT Edge).
- **Community and Support:** Availability of community support or professional services (e.g., Node-RED's active community, AWS Partner Network).

B - Why Choose a Full-Service Platform?

Full-service platforms are ideal for complex, large-scale, or enterprise-grade IoT solutions due to:

- **Comprehensive Ecosystem:** AWS and Azure offer integrated services (e.g., AWS Lambda, Azure Digital Twins) for analytics, storage, and machine learning, reducing the need for external tools (Digiteum).
- **Scalability:** They handle billions of devices and trillions of messages, suitable for industrial or commercial applications (InfiSIM).
- **Security:** Robust features like end-to-end encryption, device authentication, and compliance with standards (e.g., GDPR, HIPAA) (Matellio).
- **Managed Services:** Simplify deployment with pre-built templates (e.g., Azure IoT Central) and managed infrastructure, saving development time (Bateson, 2023).
- **Professional Support:** Access to enterprise-grade support and partner ecosystems (e.g., AWS Partner Network) (Chogale, 2024).

Choose these for projects requiring high reliability, scalability, and integration with cloud services, such as industrial IoT or smart cities.

C - Why Choose an Open-Source Approach?

Open-source platforms are preferred for flexibility, cost, and customization:

- **Cost-Effectiveness:** Free to use, reducing expenses compared to AWS's message-based pricing (e.g., \$136,000/month for 10,000 devices vs. \$3,300 for open-source on AWS infrastructure) (IoT For All).
- **Customization:** Open-source code allows tailoring to specific needs (e.g., Node-RED's visual programming, ESPHome's firmware customization for ESP devices) (Miguel, 2025).
- **Community Support:** Active communities provide resources and plugins (e.g., Node-RED's flow-sharing community, Zigbee2MQTT's device compatibility) (Rupareliya, 2025).
- **Lightweight Protocols:** Support for MQTT and CoAP, ideal for low-power devices (e.g., Zigbee2MQTT for Zigbee-based home automation) (IoT For All).
- **Local Control:** Platforms like ESPHome and Zigbee2MQTT enable local processing, reducing cloud dependency and latency for home automation (Miguel, 2025).

Choose these for small-scale, DIY, or home automation projects where cost and customization are priorities.

II. Choosing IoT Development Platforms

A - Smart Home

- **Recommended Platform:** Home Assistant or Node-RED
- **Reason:**
 - Home Assistant is open-source, highly customizable, and supports a wide range of devices (Zigbee, Z-Wave, Wi-Fi) via integrations like Zigbee2MQTT.
 - It runs locally, ensuring privacy and low latency for home automation (e.g., lights, thermostats).
 - Node-RED is ideal for rapid prototyping with its visual flow-based programming, integrating diverse devices via MQTT.
 - Both have strong community support, making them suitable for home users (Miguel, 2025).

B - Melbourne Public Transport Contactless Payment System

- **Recommended Platform:** AWS IoT Core
- **Reason:**

- A public transport system requires high scalability, security, and reliability to handle millions of transactions across numerous devices (e.g., card readers).
- AWS IoT Core supports billions of devices, offers robust security (Device Defender), and integrates with AWS services like Lambda and Kinesis for real-time transaction processing and analytics.
- Its serverless architecture ensures scalability for peak loads (Matellio; Chogale, 2025).

C - Smartwatch Health Monitoring System

- Recommended Platform: Azure IoT Central
- Reason:
 - Health monitoring systems need secure data handling, real-time analytics, and integration with healthcare standards (e.g., FHIR).
 - Azure IoT Central provides pre-built templates for rapid development, integrates with Azure Stream Analytics for real-time health data processing, and offers Azure Sphere for device security.
 - Its scalability supports millions of wearables, and Microsoft's healthcare focus ensures compliance (Muhammed & Ucuz, 2020; Matellio).

III. Scaling IoT Development Platforms

A - Smart Home (Increasing Number of Homes)

- Device Management: Scale device provisioning and monitoring for thousands of homes (e.g., Home Assistant's cloud integration for remote management).
- Data Processing: Handle increased sensor data (e.g., temperature, motion) with local hubs or cloud-based analytics (e.g., Node-RED with cloud storage).
- Network Bandwidth: Optimize for low-bandwidth protocols (e.g., MQTT) to manage multiple homes efficiently (Dayarathna, 2019).
- Security: Ensure encrypted communication and user authentication across homes to prevent breaches.
- User Interface: Scale dashboards for centralized control of multiple homes (e.g., Home Assistant's Lovelace UI).

B - Public Transport Contactless Payment System

- Transaction Throughput: Scale to process millions of contactless payments daily, requiring high-throughput message handling (e.g., AWS IoT Core’s rules engine) (infiSIM).
- Device Scalability: Support thousands of payment terminals across stations, with OTA updates for maintenance (Rishabh Software).
- Security: Implement robust encryption and fraud detection to protect financial data (Matellio).
- Latency: Ensure low-latency processing for real-time payments, possibly using edge computing (e.g., AWS IoT Greengrass) (Bateson, 2023).
- Redundancy: Use failover across regions to ensure uptime during peak travel times (Miguel, 2025).

C - Smartwatch Health Monitoring System

- Data Volume: Handle continuous health data (e.g., heart rate, SpO2) from millions of devices, requiring scalable storage (e.g., Azure Data Lake) (Rishabh Software).
- Real-Time Analytics: Scale analytics for real-time health alerts (e.g., Azure Stream Analytics for anomaly detection) (Chogale, 2024).
- Security and Compliance: Ensure HIPAA-compliant encryption and secure device authentication (Matellio).
- Device Management: Scale OTA updates and monitoring for wearable devices (Marcuta & MoldStud, 2025)
- Integration: Support integration with healthcare systems for data sharing (e.g., Azure IoT Central’s REST API) (Miguel, 2025).

IV. Comparison Matrix

Platform	AWS IoT Core	Azure IoT Central	Node-RED	Home Assistant	Zigbee2MQTT
Scalability	Billions of devices	Billions of devices	Limited, cloud-dependent	Moderate, home-scale	Small-scale, home-focused

Device Management	Robust (Device Management, OTA updates)	IoT Hub, device twins	Basic, flow-based	Local device control, integrations	Zigbee device management
Security	Device Defender, encryption	Azure Sphere, Defender	Basic, relies on host	Basic, local encryption	Basic, local security
Integration	SDKs (C, Python, Java), AWS services	.NET, Java, Python, Microsoft tools	JavaScript, REST, MQTT	Zigbee, Z-Wave, MQTT	MQTT, Zigbee
Analytics	IoT Analytics, real-time	Stream Analytics, predictive	Limited, external tools	External analytics	None, external tools
Cost	Free tier (250K msg/mo), then per-message	Free tier (8K msg/day), per-device	Free (open-source)	Free (open-source)	Free (open-source)
Ease of Use	Tutorials, complex for beginners	Drag-and-drop, templates	Visual programming, easy	User-friendly UI	Moderate, technical setup

Protocol Support	MQTT, HTTP, LoRaWAN	MQTT, AMQP, HTTP	MQTT, HTTP, CoAP	MQTT, Zigbee, Z-Wave	Zigbee, MQTT
Edge Computing	Greengrass for edge	IoT Edge, Percept	Limited, via cloud	Local processing	Local processing
Community/Support	AWS Partner Network	Microsoft support	Active community	Strong community	Active community

V. References

31west. IoT Platforms Comparison. <https://www.31west.net/blog/iot-platforms-comparison-aws-azure-google-ibm-cisco/>

AltexSoft. Making Sense of IoT Platforms: AWS vs Azure vs Google vs IBM vs Cisco. <https://www.altexsoft.com/blog/iot-platforms/>

AWS IoT Core Documentation. <https://aws.amazon.com/iot-core/> (for official AWS IoT details)

Azure IoT Central Documentation. <https://azure.microsoft.com/en-us/services/iot-central/> (for official Azure IoT details)

Bateson, C., 2023. Medium. Azure IoT vs. AWS IoT: How To Select Best Platform in 2023. <https://chrisbateson80.medium.com/azure-iot-vs-aws-iot-how-to-select-best-platform-in-2023-bfbc5021623f>

Chogale, N., 2024. Code-b. Best IOT Cloud Services Compared (Azure, GCP, AWS etc). <https://code-b.dev/blog/best-iot-cloud-services>

Dayarathna, M., 2019. DZone. Comparing 11 IoT Development Platforms.

<https://dzone.com/articles/iot-software-platform-comparison>

Digiteum: Comparing Top IoT Development Platforms. <https://www.digiteum.com/iot-platforms-comparison>

Dziuba, A., 2024. Relevant Software: AWS vs. Azure for IoT.

<https://relevant.software/blog/aws-vs-azure-for-iot/>

Hasan, M., 2022. IoT Analytics. The IoT Cloud: Microsoft Azure vs. AWS vs. Google Cloud.

<https://iot-analytics.com/iot-cloud/>

Home Assistant Official Website. <https://www.home-assistant.io/> (for Home Assistant features and community support)

InfiSIM. IoT Cloud Platforms: AWS IoT vs. Azure IoT. <https://infisim.com/blog/aws-iot-vs-azure-iot>

IoT For All: How to Choose Your IoT Platform - Should You Go Open-Source?

<https://www.iotforall.com/iot-platform-open-source-vs-serverless>

Jangid, K., 2023. Dynamics Square. Best IoT Platforms & Tools with Comparison 2024.

<https://www.dynamicssquare.co.uk/blog/best-iot-platforms/>

Marcuta, C., & MoldStud Research Team. MoldStud. <https://moldstud.com/articles/p-2025-update-comprehensive-feature-comparison-of-top-iot-development-platforms>

Matellio. Top 33 IoT Platforms to Watch for in 2025. <https://www.matellio.com/blog/top-iot-platforms/>

Miguel, P., G., 2025. The CTO Club. 30 Best IoT Cloud Platforms Reviewed in 2025.

<https://thectoclub.com/tools/best-iot-cloud-platform/>

Muhammed, A, S., Ucuz, D., 2020. Comparison of the IoT Platform Vendors, Microsoft Azure, Amazon Web Services, and Google Cloud, from Users' Perspectives.

https://www.researchgate.net/publication/342185935_Comparison_of_the_IoT_Platform_Vendors_Microsoft_Azure_Amazon_Web_Services_and_Google_Cloud_from_Users'_Perspectives

Node-RED Official Website. <https://nodered.org/> (for Node-RED documentation and flows)

Rishabh Software. AWS IoT vs. Azure IoT. <https://www.rishabhsoft.com/blog/aws-iot-vs-azure-iot-comparison>

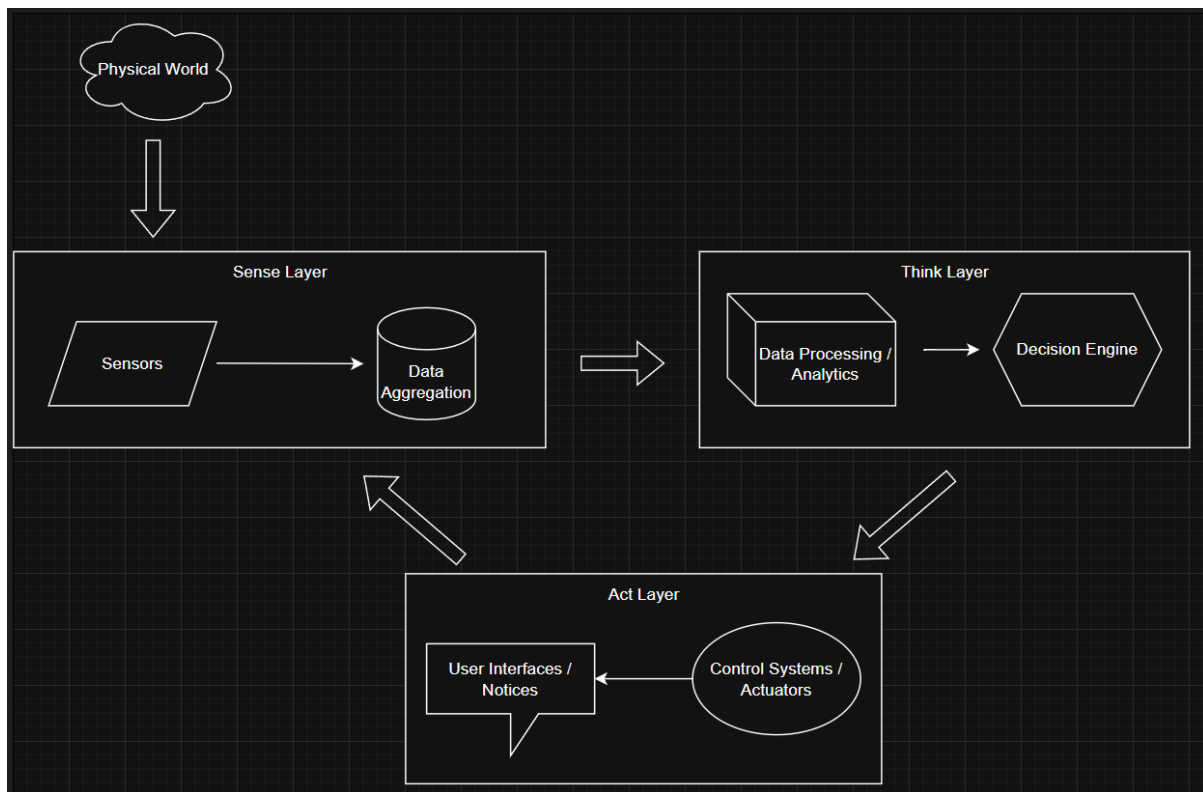
Rupareliya, K., 2025. Intuz: Top 30 IoT Development Platforms with Comparison in 2025.
<https://www.intuz.com/blog/top-iot-development-platforms-and-tools>

Sruthy, 2025. Software Testing Help. Top 10 IoT Platforms to Watch Out in 2025.
<https://www.softwaretestinghelp.com/best-iot-platforms/>

Zigbee2MQTT Official Website. <https://www.zigbee2mqtt.io/> (for Zigbee device support)

B – Group Discussions

1. Sketch a block diagram of the system.



Key components:

- Physical layer: Vehicles, pedestrians, infrastructure
- Sensing layer: Cameras, GPS, speed sensors, occupancy detectors
- Network layer: MQTT brokers, gateways
- Processing layer: Traffic analysis, prediction models
- Application layer: Traffic control, navigation systems

2. What are all the physical objects in the space (cars, people, traffic lights)?

- Vehicles: Cars, buses, trucks, bicycles, scooters, emergency vehicles
- Infrastructure: Traffic lights, streetlights, road signs, toll booths, parking meters
- People: Pedestrians, cyclists, traffic officers
- Public Transport: Trains, trams, buses, stations
- Road Elements: Lanes, intersections, crosswalks, bridges

3. What is the data that is generated from sensors (car locations, traffic)?

- Vehicle Data: GPS location, speed, direction, acceleration, occupancy
- Traffic Flow: Vehicle counts, classification (car/truck/bus), queue lengths
- Environmental: Road conditions (wet/icy), visibility, temperature
- Pedestrian: Crosswalk counters, pedestrian density
- Public Transport: Bus/train locations, arrival times, passenger counts
- Infrastructure Status: Traffic light status, parking space availability

4. What are the actuations of the system (traffic lights, car navigation, train speeds)?

- Traffic Control: Adjusting traffic light timing, dynamic lane assignment
- Navigation Systems: Rerouting suggestions to drivers/riders
- Public Transport: Adjusting train/bus frequencies and speeds
- Information Displays: Updating digital signage with traffic info
- Emergency Systems: Prioritizing emergency vehicle routes
- Parking Systems: Guiding drivers to available parking

5. What are the processes and decisions that need to be made to connect the sensors to the actuators?
 - Data Fusion: Combining data from multiple sources for accuracy
 - Traffic Prediction: Using historical and real-time data to forecast congestion
 - Priority Management: Emergency vehicle prioritization
 - Load Balancing: Distributing traffic across alternative routes
 - Anomaly Detection: Identifying accidents or unusual events
 - Demand Response: Adjusting public transport based on demand
 - Optimization Algorithms: Minimizing overall travel time
6. Plan a hierarchical MQTT application-level communications protocol. Think about the identity of users and devices in the system and plan single-level or multi-level wildcards for message filtering.
 - Topic Structure: smartcity/transport/[region]/[device_type]/[device_id]/[data_type]
 - Example topics:
 - smartcity/transport/central/trafficlight/42/status
 - smartcity/transport/east/bus/571/location
 - smartcity/transport/north/parking/zone3/occupancy
 - Wildcard Usage:
 - ❖ Single-level (+): smartcity/transport+/trafficlight+/status
 - ❖ Multi-level (#): smartcity/transport/central/#
 - Device Identity Management:
 - ❖ Each device has a unique ID and publishes to its specific topic
 - ❖ Devices subscribe to relevant control topics

- ❖ Central systems use wildcards to monitor groups of devices
- Security Considerations:
 - ❖ TLS encryption for all communications
 - ❖ Device authentication using client certificates
 - ❖ Fine-grained access control for topics

C – Technical Tasks

Node-RED Setup

```

node-red
Microsoft Windows [Version 10.0.26100.4770]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Hayden Duong>npm install -g --unsafe-perm node-red

added 278 packages in 12s

52 packages are looking for funding
  run `npm fund` for details

C:\Users\Hayden Duong>node-red
3 Aug 22:14:43 - [info]

Welcome to Node-RED
=====
3 Aug 22:14:43 - [info] Node-RED version: v4.1.0
3 Aug 22:14:43 - [info] Node.js version: v22.16.0
3 Aug 22:14:43 - [info] Windows_NT 10.0.26100 x64 LE
3 Aug 22:14:44 - [info] Loading palette nodes
3 Aug 22:14:44 - [info] Settings file   : C:\Users\Hayden Duong\.node-red\settings.js
3 Aug 22:14:44 - [info] Context store   : 'default' [module=memory]
3 Aug 22:14:44 - [info] User directory : C:\Users\Hayden Duong\.node-red
3 Aug 22:14:44 - [warn] Projects disabled : editorTheme.projects.enabled=false
3 Aug 22:14:44 - [info] Flows file      : C:\Users\Hayden Duong\.node-red\flows.json
3 Aug 22:14:44 - [info] Creating new flow file
3 Aug 22:14:44 - [warn]

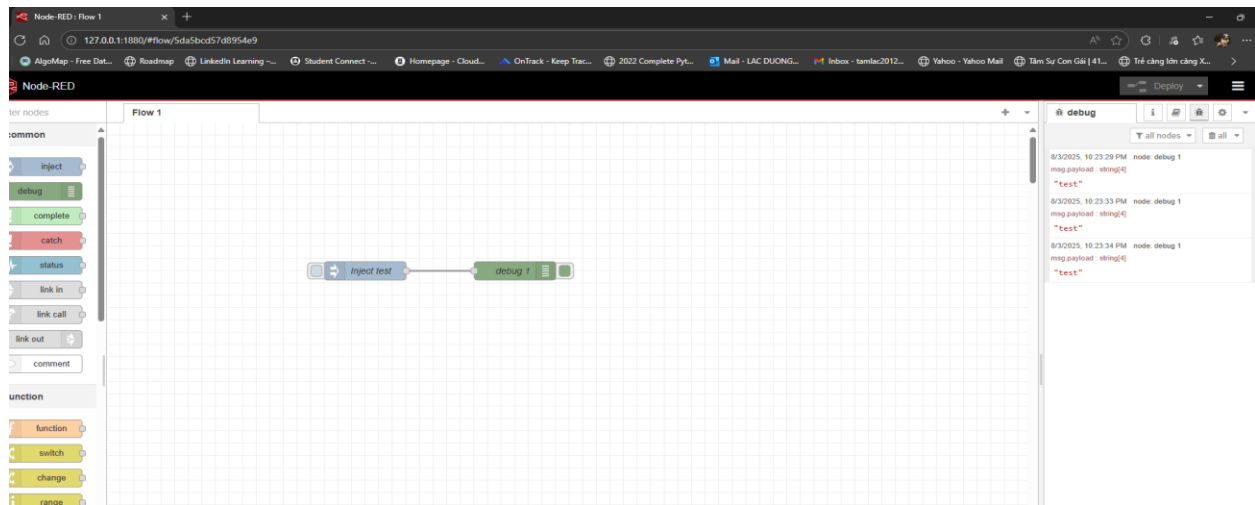
-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----
3 Aug 22:14:44 - [info] Server now running at http://127.0.0.1:1880/
3 Aug 22:14:44 - [warn] Encrypted credentials not found
3 Aug 22:14:44 - [info] Starting flows
3 Aug 22:14:44 - [info] Started flows
|
  
```

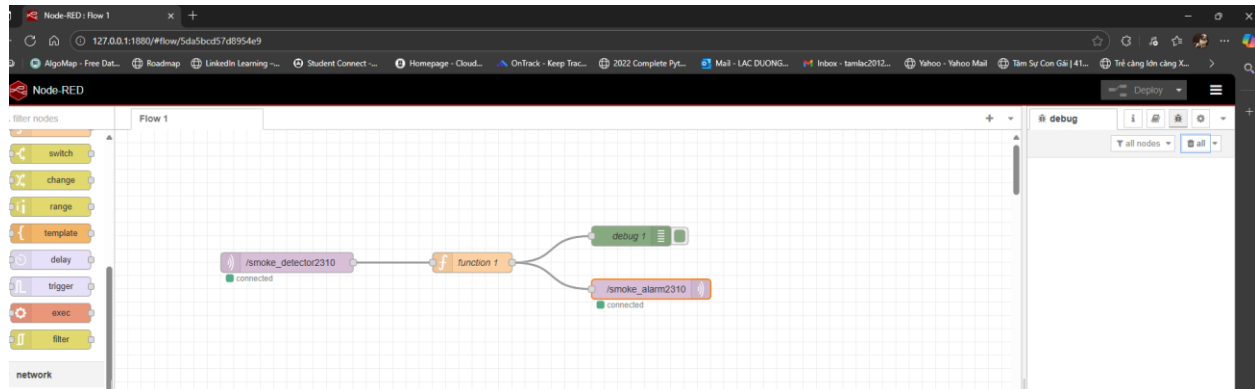
Picture 1 – Node-Red is successfully installed.

A Simple Application

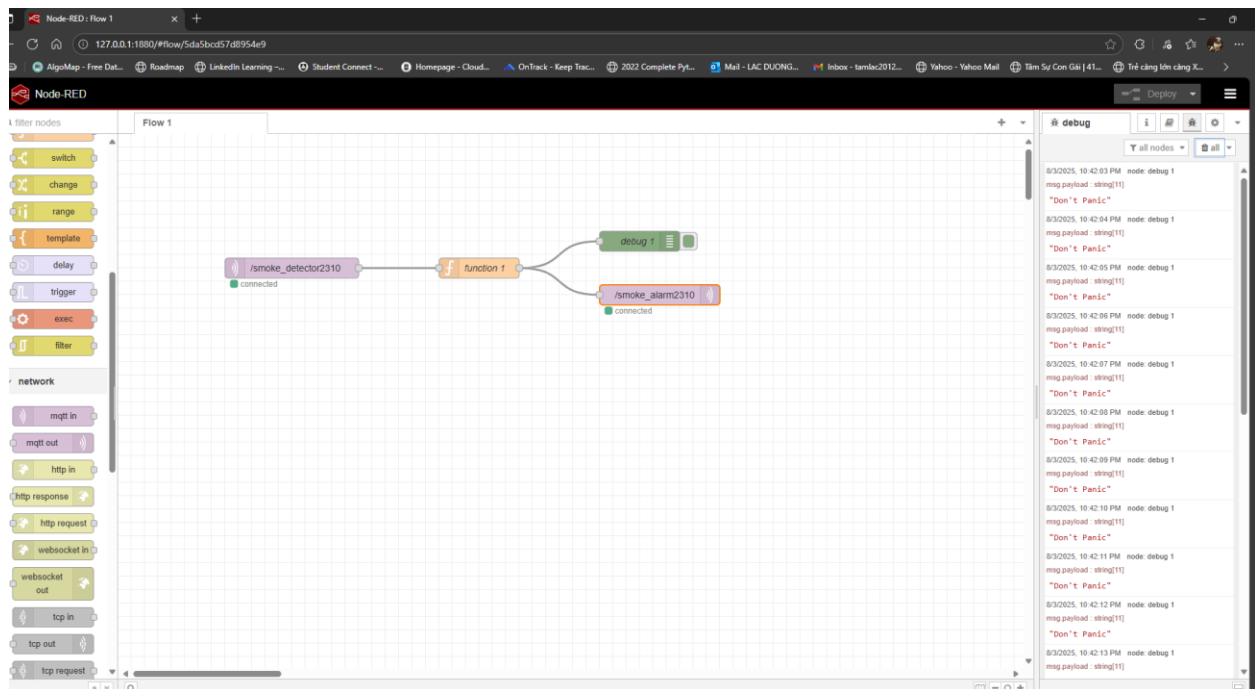


Picture 2 – Successfully sent string “test” as payloads.

Building a Smoke alarm by Combining Node-RED with MQTT



Picture 3 – Successfully deployed the system as instructed.



Picture 4 - Successfully display warning messages.

Lesson Summary

In this module, I was introduced with current IoT Connectivity Technologies, LowPAN, LPWAN, MQTT, and how to use Node-RED which are very crucial to me as I can start working on my proposed plan for D-task.

I would say this module is more of a practical one instead of theoretical heavy compared to the previous one. Nonetheless, it is important for the upcoming modules and I need to grasp it 100% as soon as possible.

